intel®

APPLICATION NOTE

AP-161

September 1983

Complex Peripheral Control with the UPI-42

NOVEMBER 1983
ORDER NUMBER: 230795-001

COMPLEX PERIPHERAL CONTROL WITH THE UPI-42

TABLE OF CONTENTS

FIGURES

230795-001

FLOW CHARTS

Addtional sources of information on Intel's UPI devices;

"UPI User's Manual"
Includes the following Application Notes;
Programmable Keyboard Interface
Using the 8295 Dot Matrix Printer Controller
An 8741A/8041A Digital Cassette Controller

"8048 Family Applications Handbook"

"1983 Microprocessor and Peripheral Handbook"

"MCS-48 and UPI-41A/42 Assembly Language Manual"

"Specifications for Impact Dot Matrix Printer Model-3210", Epson, Jan 8, 1981

INTRODUCTION

The UPI-42 is the newest member of Intel's Universal Peripheral Interface (UPI) microcomputer family. It represents a significant growth in UPI capabilities resulting in a broader spectrum of applications. The UPI-42 incorporates twice the EPROM/ROM of the UPI-41A, 2048 vs 1024 bytes, twice the RAM, 128 vs 64 bytes, and operates at a maximum speed twice that of the UPI-41A, i.e. 12 MHz vs 6 MHz. The ROM based 8042 and the EPROM based 8742 provide more highly integrated solutions for complex stepping motor and dot matrix printer applications. Those applications previously requiring a microprocessor plus a UPI chip can now be implemented entirely with the UPI-42.

The software features of the UPI-42, such as indirect Data and Program Memory addressing, two independent and selectable 8 byte register banks, and directly software testable I/O pins, greatly simplify the external interface and software flow. The software and hardware design of the UPI-42 allows a complex peripheral to be controlled with a minimum of external hardware.

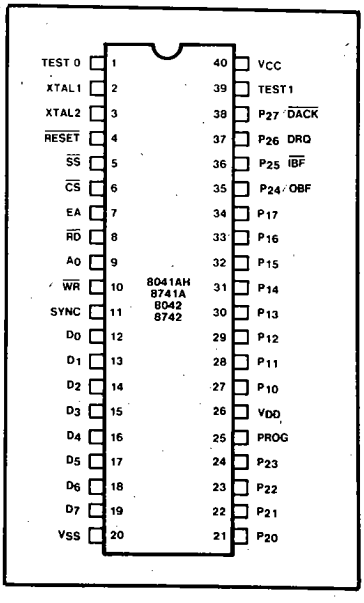


Figure 1. UPI-42 Pin Configuration

Many microcomputer systems need real time control of peripheral devices such as a printer, keyboard, complex motor control or process control. These medium speed but still time consuming tasks require a fair amount of system software overhead. This processing burden can be reduced by using a dedicated peripheral control processor

Until recently, the dedicated control processor approach was usually not cost effective due to the large number of components needed; CPU, RAM, ROM, I/O, and Timer/Counters. To help make the approach more cost effective, in 1977 Intel introduced the UPI-41 family of Universal Peripheral Interface controllers consisting of an 8041 (ROM) device and an 8741 (EPROM) device. These devices integrated the common microprocessor system functions into one 40 pin package. The UPI-42 family, consisting of the 8042 and 8742, further extends the UPI's cost effectiveness through more memory and higher speed.

Another member of the UPI family is the Intel 8243 Input/Output Expander chip. This chip provides the UPI-41A and UPI-42 with up to 16 additional independently programmable I/O lines, and interfaces directly to the UPI-41A/42. Up to seven 8243s can be cascaded to provide over 100 I/O lines.

The UPI is a single chip microcomputer with a standard microprocessor interface. The UPI's architecture, illustrated in Figure 3, features on-chip program memory, ROM (8041A/8042) or EPROM (8741A/8742), data memory (RAM), CPU, timer/counter, and I/O. Special interface registers are provided which enable the UPI to function as a peripheral to an 8-bit central processor.

Using one of the UPI devices, the designer simply codes his proprietary peripheral control algorithm into the UPI device itself, rather than into the main system software. The UPI device then performs the peripheral control task while the host processor simply issues commands and transfers data. With the proliferation of microcomputer systems, the use of UPIs or slave microprocessors to off load the main system microprocessor has become quite common.

This Application Note describes how the UPI-42 can be used to control dot matrix printing and the printer mechanism, using stepper motors for carriage/print head assembly and paper feed motion. Previous Intel Application Notes AP-27, AP-54, and AP-91 describe using intelligent processors and peripherals to control single solenoid driven printer mechanisms with 80 character line buffering and bidirectional printing. This Application Note expands on these previous themes and extends the concept of complex device control by incorporating full 80 character line buffering, bidirectional printing, as well as drive and feedback control of two four phase stepper motors.

The Application Note assumes that the reader is familiar with the 8042/8742 and 8243 Data Sheets, and UPI-41A/42 Assembly Language. Although some background information is included, it also assumes a basic understanding of stepper motors and dot matrix printer mechanisms. A complete software listing is included in Appendix A.

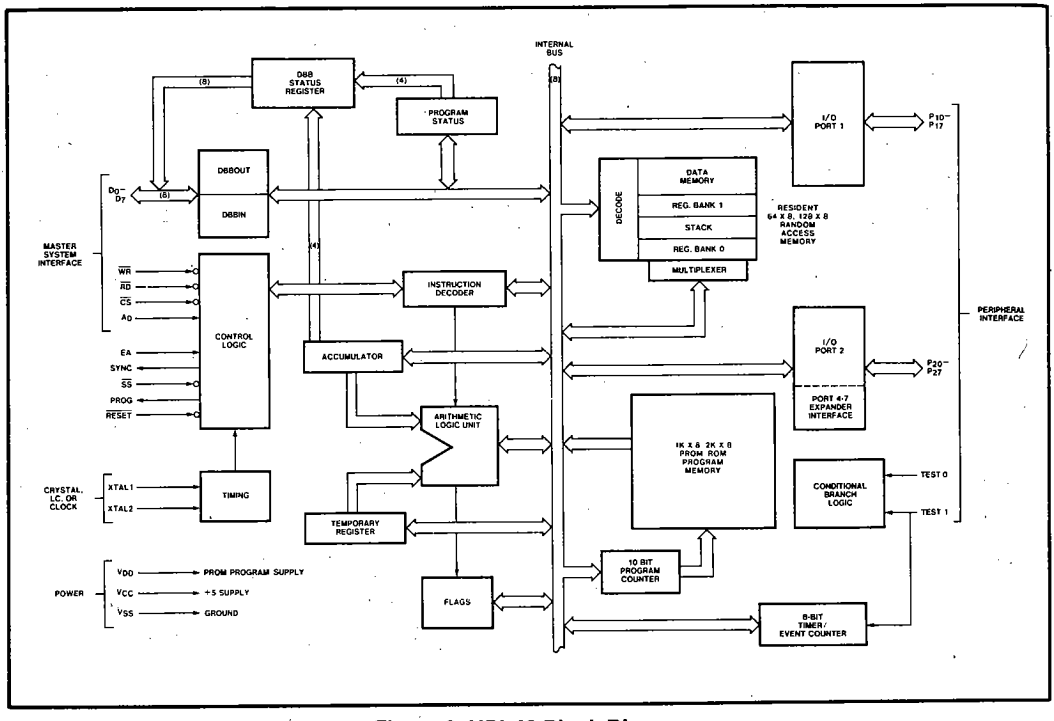


Figure 2. UPI-42 Block Diagram

DOT MATRIX PRINTING

A dot matrix printer print head typically consists of seven to nine solenoids, each of which drives a stiff wire, or hammer, to impact the paper through an inked ribbon. Characters are formed by firing the solenoids to form a matrix of "dots" (impacts of the wires). Figure 4 shows how the character "E" is formed using a 5 x 7 matrix. The columns are labeled C1 through C5, and the rows R1 through R7. The print head moves left-to-right across the paper, so that at time T1 the head is over column C1. The character is formed by activating the proper solenoids as the print head sweeps across the character position.

Dot matrix printers are a cost effective way of providing good quality hard copy output for microcomputer systems. There is an ever increasing demand for the moderately priced printer to provide more functionality with improved cost and performance. Using stepper motors to control the paper feed and carriage/print head assembly motion is one way of enabling the dot matrix printer to provide more capabilities, such as expanded or contracted characters, dot or line graphics, variable line and character spacing, and subscript or superscript printing.

However, stepper motors require fairly complex contol algorithms. Previous solutions involved the use of a main CPU, UPI, RAM, ROM, and I/O onboard the peripheral. The CPU acted as supervisor and used parallel processing to achieve accurate stepper motor control via a UPI, character buffering via the I/O device, RAM, and ROM. The CPU performed real-time decoding of each character into a dot matrix pattern. This Application Note demonstrates that the increased memory and performance of the UPI-42 facilitates integrating these control functions to reduce the cost and component count.

THE PRINTER MECHANISM

The printer mechanism used in this application is the Epson Model 3210. It consists of four basic subassemblies; the chassis or frame, the paper feed mechanism and stepper motor, the carriage motion mechanism and stepper motor, and the print head assembly.

The paper feed mechanism is a tractor feed type. It accomodates up to 8.5 inch wide paper (not including tractor feed portion). There is no platen as such; the paper is moved through the paper guide by two sprocketed wheels mounted on a center sprocket shaft. The sprocket shaft is driven by a four phase stepper motor. The rotation of the stepper motor is transmitted to the sprocket shaft through a series of four reduction gears.

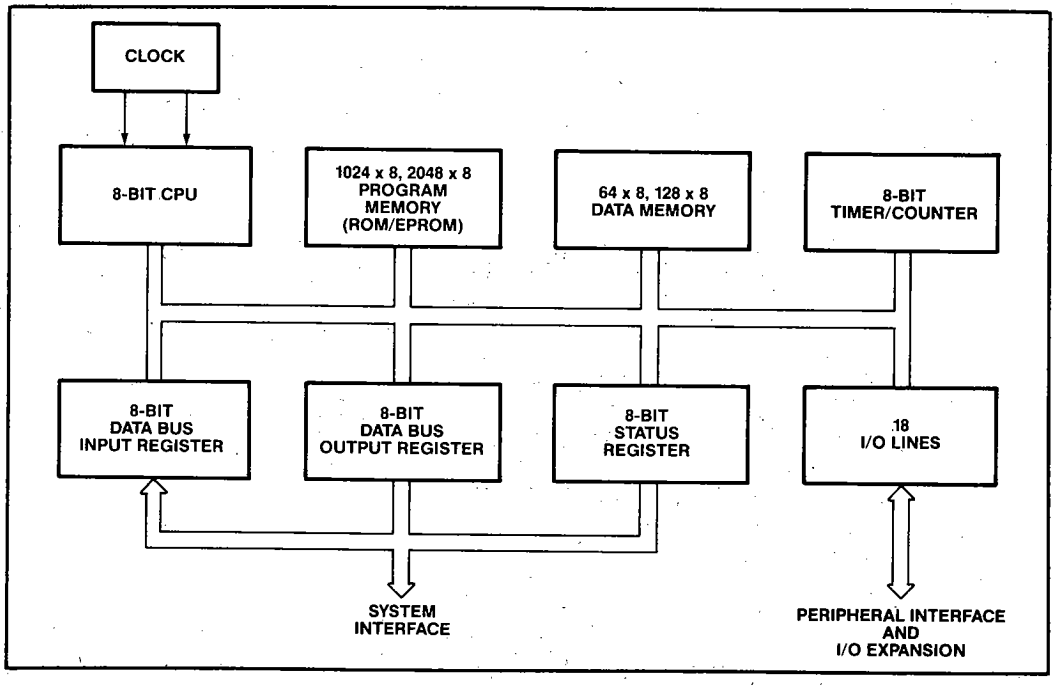


Figure 3. UPI-41A, 42 Functional Block Diagram

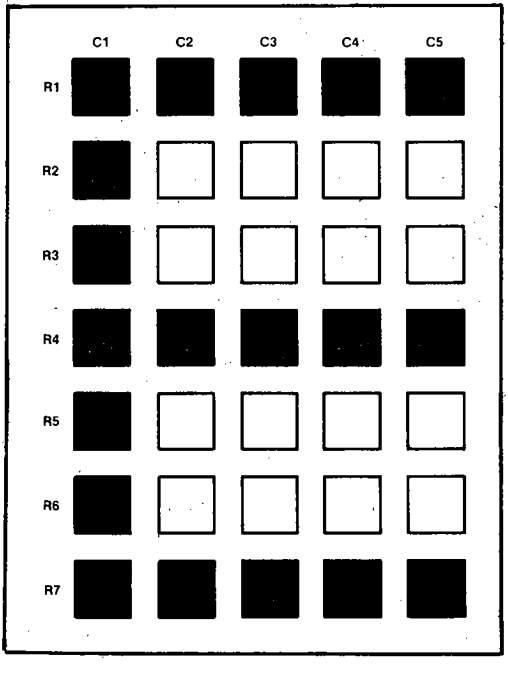


Figure 4. Character E in 5 x 7 Dot Matrix Format

The carriage motion mechanism consists of another four phase stepper motor which controls the left-to-right or right-to-left print head assembly motion. The print speed is 80 CPS maximum. Both the speed of the stepper motor and the movement of the print head assembly are independently controllable in either direction. The rotation of the stepper motor is converted to the linear motion of the print head assembly via a series of reduction gears and a toothed drive belt. The drive belt also controls a second set of reduction gears which advances the print ribbon as the print head assembly moves.

Two optical sensors provide feedback information on the carriage assembly position and speed. The first of these optical sensors, called the 'HOME RESET' or HR, is mounted near the left-most physical position to which the print head assembly can move. As the print head assembly approaches the left-most position, a flange on the print head assembly interferes with the light source and sensor, causing the output of the sensor to shift from a logic level one to zero. The right-most printer position is monitored in software rather than by another optical sensor. The right-most print position is a function of the number of characters printed and the distance required to print them.

The second optical sensor, called the 'PRINT TIMING SIGNAL' or PTS, provides feedback on carriage stepper motor velocity and relative position within a

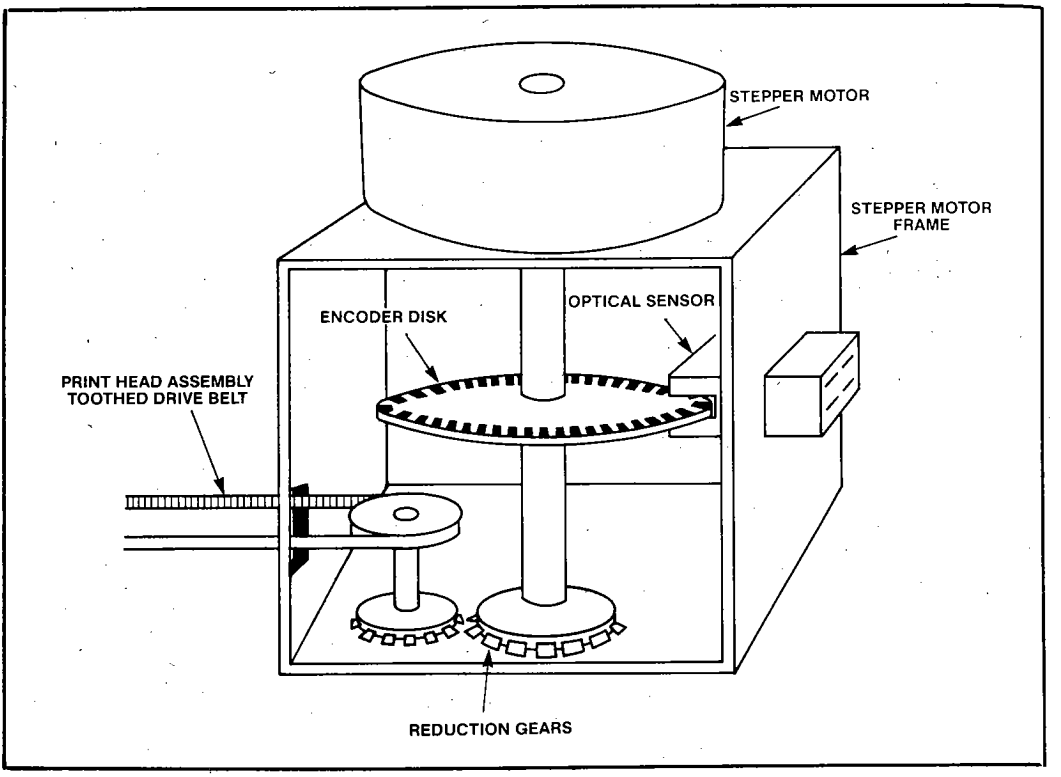


Figure 5. Carriage Stepper Motor Assembly

given step of the motor. The feedback is generated by the optical sensor as an "encoder disk" moves across it. Figure 5 illustrates the carriage stepper motor, optical sensor, encoder disk and reduction gears, and drive belt assembly. The optical sensor outputs a pulse train with the same period as the phase shift signal used to drive the stepper, but slightly out of phase with it when the motor is at a constant speed (see Software Functional Block: Phase Shift Data for additional details). The disk acts as a timing wheel, providing feedback to the UPI software of the carriage speed, position, and optimum position for energizing the print head solenoids. The two optical sensors are monitored under software and provide the critical feedback needed to control the print head assembly and paper feed motion accurately. The process of stepper motor drive and control via feedback signals is called "closed loop" stepper motor control, and is covered in more detail in the software discussion.

The print head assembly consists of nine solenoids and nine wires or hammers. Figure 6 illustrates a print head assembly. The available dot matrix measures 9 x 9. This large matrix enables the Epson 3210 print mechanism to print a variety of character fonts, such as expanded or contracted characters, as well as line or block graphics (see Appendix B, Printer Enhancements). It also facilitates printing lower case ASCII characters with "lower case descenders." That is to say, certain lower case letters (e.g. y, p, etc.) will print below the bottom part of all upper case letters.

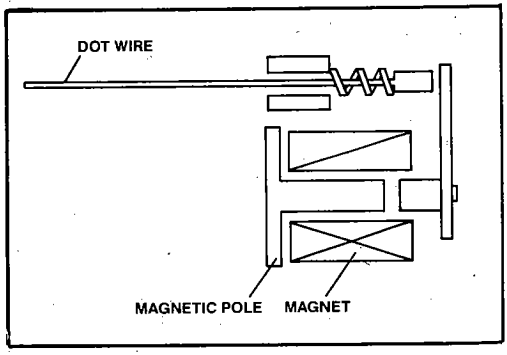


Figure 6. Print Head Solenoid Assembly

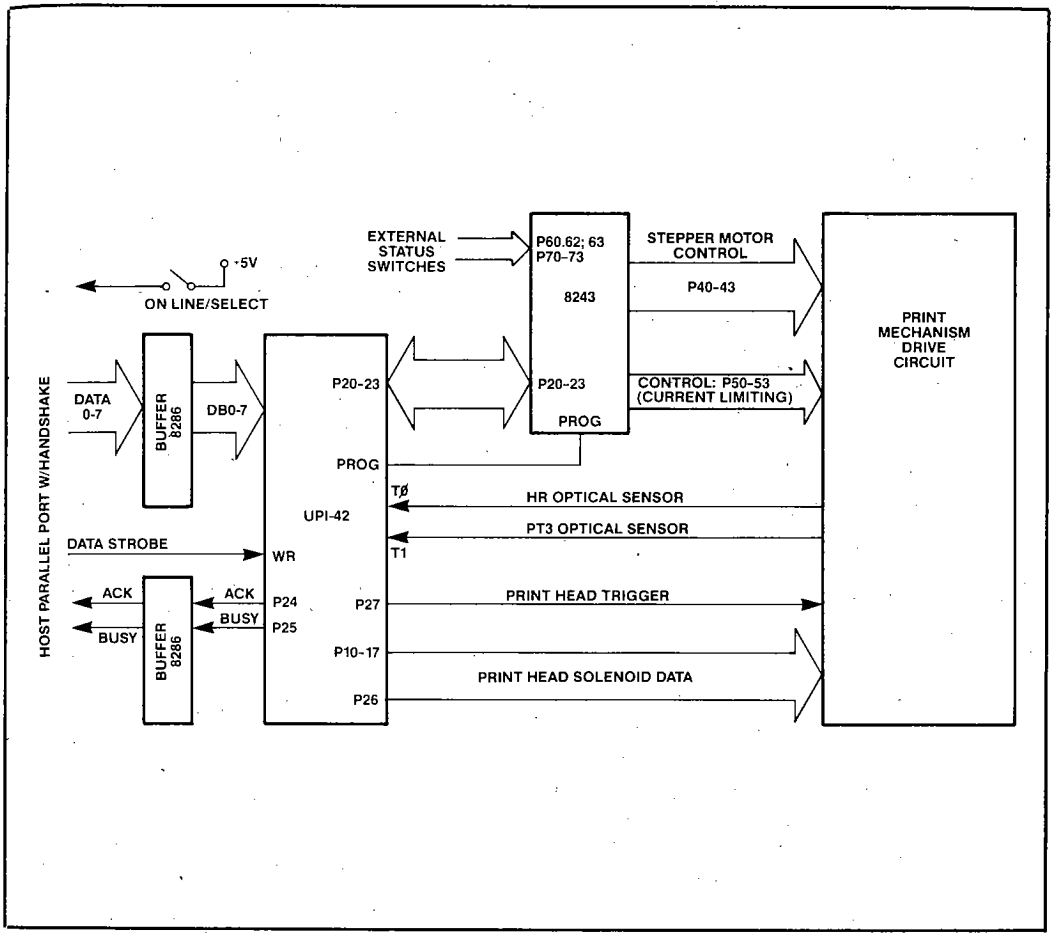


Figure 7. Hardware Interface Block Diagram

HARDWARE DESCRIPTION

Figure 7 shows a block diagram of the UPI-42 and 8243 interface to the printer mechanism drive circuit. A complete schematic is shown in Figure 8. The UPI-42 provides all signals necessary to control character buffering and handshaking, paperfeed and carriage motion stepper motor timing, print head solenoid activation, and monitoring of external status switches.

The Epson 3210 printer mechanism manual recommends a specific interface circuit to provide proper drive levels to the stepper motors windings and print head solenoids. The hardware interface used for this Application Note followed those recomendations exactly (see Appendix C, Printer Mechanism Drive Circuit Schematics).

I/O Ports

The lower half of the UPI-42 Port 2, pins 0-3, provides an interface to the 8243 I/O expander. The PROG pin of the UPI-42 is used as a strobe to clock address and data information via the Port 2 interface. The extra 16 I/O lines of the 8243 become PORTS 4, 5, 6, and 7 to the UPI software. Combined, the UPI-42 and 8243 provide a total of 28 independently programmable I/O line. These lines are used as follows:

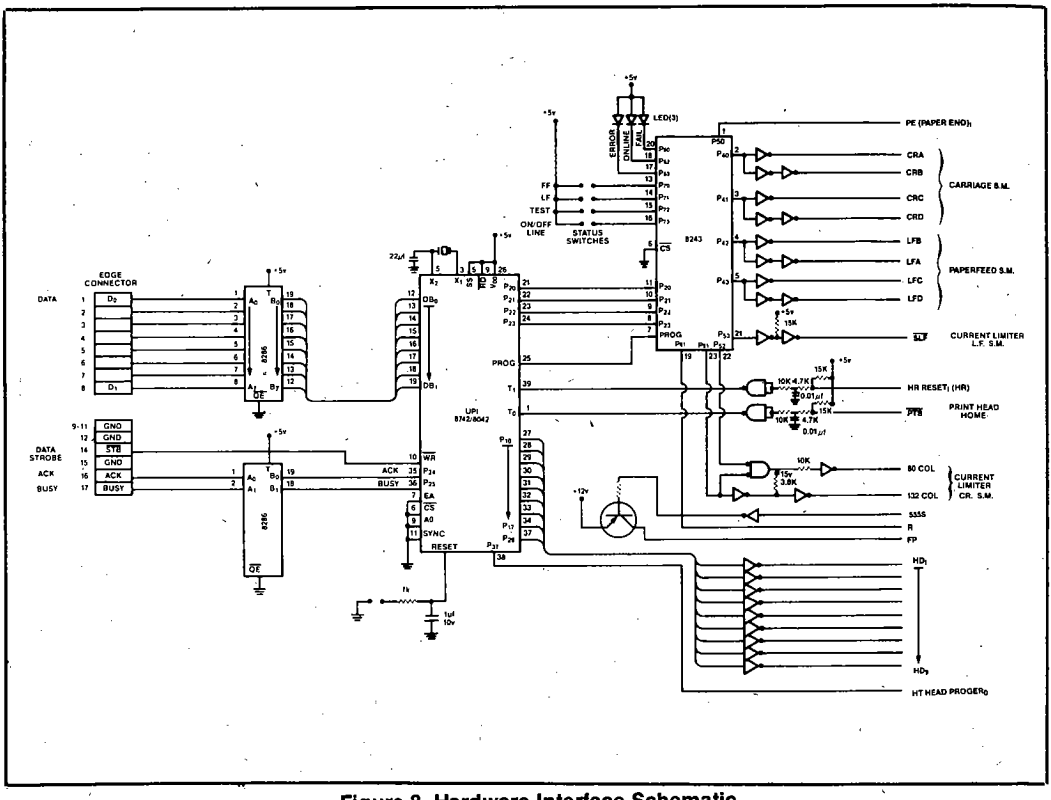


Figure 8. Hardware Interface Schematic

Port	No of lines	Bits	I/O	Description
1	8	0-7	O	Character dot column data to print head solenoids
2	1	6	O	(same)
2	1	7	O	Print head solenoid trigger
2	2	4,5	O	Host system data transfer handshaking (ACK/BUSY)
4	4	0-3	O	Carriage & paper feed stepper motors
5	3	1-3	O	Stepper motor select and current limiting
5	1	0	I	Paper End sense
6	1	1	O	Print head trigger reset
6	3	0,2,3	-	(unused)
7	5	0-3	I	External status switches; (LF, FF, TEST, ON/OFF Line)

Figure 9. UPI-42 and 8243 I/O Port Map

Note: The notation used in the balance of this Application Note, when referring to a port number and a particular pin or bit, is Port 23 rather than Port 2 bit 3.

The two printer mechanism optical sensors, discussed in the Printer Mechanism discription, are tied to the two "Test Input" pins, T0 and T1, of the UPI-42 through a buffer circuit for noise supression. These inputs are directly testable in software.

Host System Interface

The host system interfaces to the printer through a parallel port to the UPI-42 Data Bus. Four handshaking signals are used to control data transfer; Data Strobe (STB/), Acknowledge (ACK), Busy (BUSY), and Online or Select. The Data Strobe line of the host parallel port is tied directly to the UPI-42 WR/ pin. This provides a low going pulse on the UPI-42 WR/ pin whenever a data byte is written to the UPI-42. The ACK and BUSY handshake signals are tied to two UPI-42 I/O port lines for software control of data transfer. The "On Line" handshake signal is tied to a single-pole single-throw fixed position switch, which externally enables or disables character transfer from the host system. Characters transmitted to the UPI-42 by the host are loaded into the UPI-42 Data Bus Buffer In (DBBIN) register, and the Input Buffer Full (IBF) interrupt and UPI-42 status flag are set (see Figure 9. UPI-42 and 8243 I/O Ports).

Stepper Motor Interface

Port 4 (41-43) of the 8243, provides both carriage and paper feed stepper motor phase shift signals to the printer mechanism drive circuit. Each of the two stepper motors is driven by 2 two phase excitation signals (4 phases). Figure 10 shows the wave form for each stepper motor. Each signal consists of two components (Sig. 1 A/B & Sig. 2 C/D) 180 degrees out of phase with the other. Each of these signal pairs (A/B & C/D) is 90 degrees out of phase with the other pair. For each signal pair, one port line supplies both halves by using an inverter.

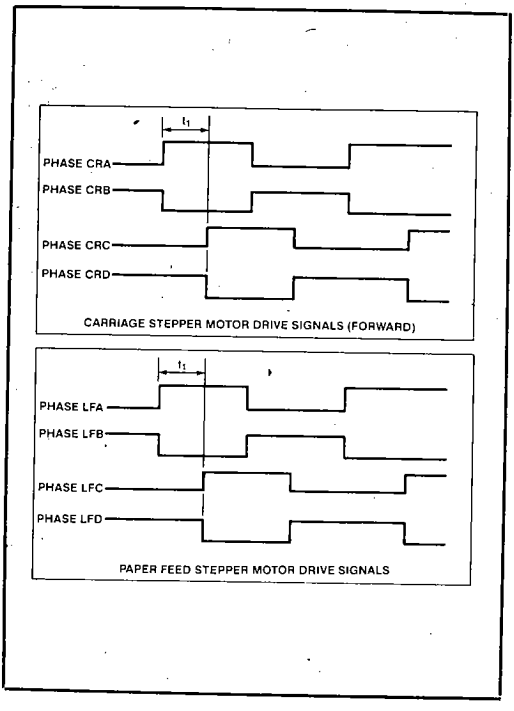


Figure 10. Stepper Motor Step Sequence Waveforms

Each of the resulting eight stepper motor drive signals is interfaced to a discrete drive transistor through an inverter. The emitter of the drive transistor is tied to the open collector of the inverter to provide high current sinking capability for the drive transistor. Each half of the motor winding is tied to the collector of the drive transistor (see Appendix C, Printer Mechanism Drive Circuit Schematic).

Each stepper motor requires two current levels for operation. These levels are called "Rush" current and "Hold" current. Rush current refers to the high current required to cause the rotor to rotate within its windings as the polarity of the power applied to the windings is changing. Each change in the polarity of the power applied to the motor windings is called a step or phase shift. Hold current refers to the low level of current required to stabilize and maintain the rotor in a fixed position when the the polarity applied to the windings is not changing. Hold current is simply Rush current with a current limiting transistor switched in. Switching from Hold to Rush current "selects" or enables that stepper motor to move with the next step signal output. In the balance of this Application Note, the term "select" will be used to refer to turning on Rush current, and "deselect" will refer to switching to Hold currrent.

Three 8243 port lines are dedicated to the select/deselect control of the two stepper motors. One line is for the paper feed stepper motor, and two lines are for the carriage motion stepper motor (80 and 132 column). These lines are labeled SLF, 80Col, and 132Col, and are 8243 PORT 53, 52, and 51, respectively.

By varying the voltage applied to the stepper motor biasing circuit and the current, it is possible to vary the distance the motor moves the print head assembly with each step. Enabling one of two different voltage biasing levels, and changing the timing rate at which the motor is stepped, facilitates either 80 or 132 character column printing. Only 80 character column printing is implemented in the software design. Appendix B, Printer Enhancements, details the software algorithm for handling 132 character printing.

Print Head Interface

A total of eleven I/O lines are used to control the print head solenoids and solenoid firing (see Figure 9 above). Nine are used for character dot data, one for the Print Head Trigger, and one for Reset of the Print Head Trigger circuit. Each of the nine character dot data lines is buffered by an open collector hex inverter.

The Print Head Trigger output is tied to the Trigger input of a 555 Monostable Multivibrator. The output pulse generated by the 555 triggers the print head solenoids to fire. The 555 Output pulse width is independent of the input trigger waveform. The pulse width is determined by an RC network across the 555 inputs and the voltage level applied to the Control Voltage 555 input. The 555 Output is tied to the base of a PNP transistor through an inverter, biased in a normally off configuration. The PNP transistor supplies enough drive to pull up the open collector inverter on each print head solenoid line, Port 10-17 and 26. The 555 output pulse momentarily enables the print head solenoid line open collector inverter output, turning on the solenoid drive transistor, and firing the print head hammer. The 555 Ouput pulse width is approximately 400 us. Further details of the print head firing operation can be found in the software description below.

Miscellaneous Interface Signals

The 8243 Port 5 pin 0 is tied to the Paper End Detector, a reed switch located on the printer paper guide. This sensor detects when the paper is nearly exhausted.

Three LED status lights complete the hardware interface design. One status light is used for each of the following: Power ON/OFF, On/Off Line, and Out of Paper.

BACKGROUND

Before a detailed discussion of the software begins, a few terms and software functions referenced throughout the software need introduction.

A. What is a Stepper Motor?

A stepper motor has the ability to rotate in either direction as well as start and stop at predetermined angular positions. The stepper motor's shaft (rotor) moves in precise angular increments for each input step. The displacement is repeated for each input step command, accurately positioning the rotor for a given number and sequence of steps.

The stepper motor controls position, velocity, and direction. The accuracy of stepper motors is generally 5 percent of one step. The number of steps in each revolution of the shaft varies, depending on the intended application.

B. Open/Closed Loop Stepper Motor Drive and Control

The carriage stepper motor is closed loop driven. The paper feed stepper motor is open loop driven.

There are two major types of stepper motor control known by the broad headings of open and closed loop. Open loop is simply continuous pulses to drive the motor at a predetermined rate based on the voltage, current, and the timing of the step pulses applied. Closed loop control is characterized by continuous monitoring of the stepper motor, through feedback signals, and adjusting the motor's operation based upon the feedback received.

C. Stepper Motor Drive Phase Shift or Step Sequence

Each change in the polarity of the power applied to the motor windings is called a step or phase shift. The sequence of the steps or phase shifts, and the pattern of polarity changes output to the stepper motor, determines the direction of rotation.

Figure 10 shows the waveforms for each of the two stepper motors. Figure 11 lists the step sequence for carriage motor clockwise rotation, which moves the print head assembly Left-to-Right. Figure 11 also lists the step sequence for counterclockwise rotations; the print head assembly moves Right-to-Left. Figure 12 lists the step sequence for the paper feed stepper motor clockwise drive. The phase sequence, for either stepper motor, may begin at any point within the sequence list, but must then continue in order.

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	Off	On
2	On	Off	On	Off
3	Off	On	On	Off
4	Off	On	Off	On

Carriage stepper motor rotates clockwise
Print head assembly moves from left to right

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Carriage stepper motor rotates counter clockwise
Print head assembly moves from right to left

Figure 11. Carriage Stepper Motor Step Sequence

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Figure 12. Paper Feed Stepper Motor Step Sequence

C. Acceleration and Deceleration of Stepper Motors

The carriage stepper motor starts from a fixed position, accelerates to a constant speed, maintains constant speed, and then decelerates to a fixed position. Printing may occur from the time and position the print head assembly reaches constant speed, until the time and position the print head assembly begins to decelerate from constant speed. Whether printing occurs during any carriage stepper motor drive sequence is controlled by software. Figure 18, below, illustrates these components of print head assembly line motion.

Due to inertia, a finite time interval and angular displacement is required to accelerate a stepper motor to its full speed. Conversely, deceleration must begin some time before the final angular position. The time interval and angular displacement of the carriage stepper motor translates into the distance the print head assembly travels before it reaches a constant speed. The distance traveled during acceleration is constant. The distance the print head assembly travels during deceleration must be the same as the distance traveled during acceleration in order to accurately align the character dot columns from one line to the next.

E. Stepper Motor Predetermined Time Constant

Whenever the stepper motor is stepped, or energized, the angular velocity of the rotor is greater than the constant speed which is ultimately required. This is called "overshoot." The frictional load of the carriage assembly (motor rotor, reduction gears, drive belt and print head assembly, or paper feed sprocket shaft and wheels) provides damping or frictional load. Damping slows the motor to less than the required constant speed and is called "undershoot" (see Figure 13, Carriage Stepper Motor Drive Timing). A constant rate of speed is achieved through the averaging of the overshoot and undershoot within each step.

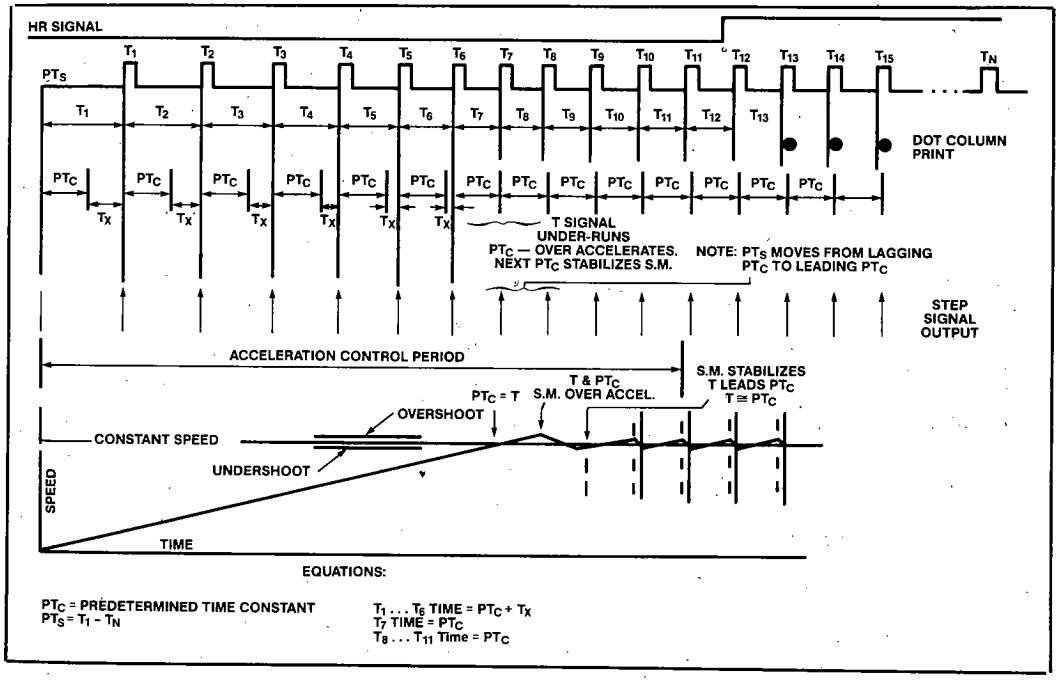


Figure 13. Carriage Stepper Motor Drive Timing

The Predetermined Time (PT) Constant is the time required to average the overshoot and undershoot of the particular stepper motor for a desired constant rate of speed. The PT also is the time required to move the print head assembly a specific distance, acounting for both overshoot and undershoot of the stepper motor.

Changing the Predetermined Time Constant changes the angular displacement of the stepper motor rotor, this in turn changes the output. Figure 14 lists the Time Constants for both standard and condensed character printing. Figure 15 lists the paper feed stepper motor Time Constants used for various line spacing formats. This Application Note implements standard character print and paper feed (6 lines per inch) Time Constants. See Appendix B, Printer Enhancements, for details on implementing non-standard Time Constants.

Character mode	Predetermined time	
Standard or Enlarged Character	2.08ms	+10% -4%
Condensed Character	4.16ms	+10% -4%

Figure 14. Carriage Stepper Motor Predetermined Time Constants

Paper feed pitch

0.12mm(1/216") /1 pulse
4.23mm(1/6") /36 pulses
3.18mm(1/8") /27 pulses
2.82mm(1/9") /24 pulses

Paper feed time

150ms/4.23mm	Approx. 6.6 lines/s (continuous feed)
113ms/3.18mm	Approx. 8.8 lines/s (continuous feed)
100ms/2.82mm	Approx. 10 lines/s (continuous feed)

Figure 15. Paper Feed Stepper Motor Predetermined Time Constants

D. Relationship Between PTS and PT

Figure 13 illustrates how PTS lags PT at the start of acceleration, and moves to lead PT as the motor achieves constant speed. Figure 13 also illustrates the relationship between HR, PTS, PT, acceleration, constant speed, and printing. Figure 16 and 17 illustrate the relationship between PTS and PT during acceleration and at constant speed.

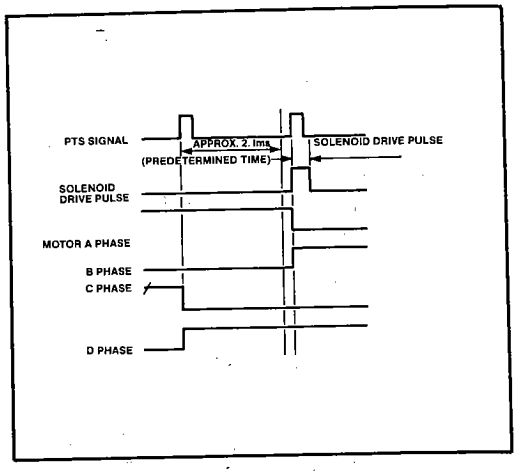


Figure 16. PTS Lags PT Timing

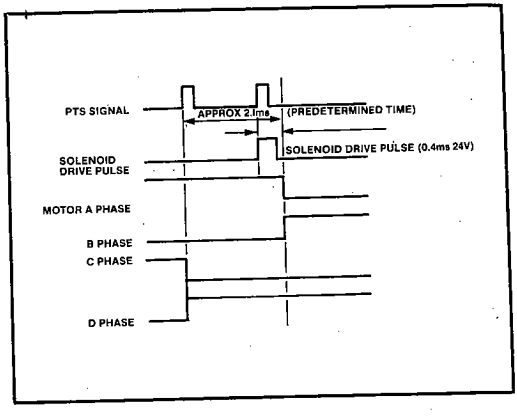


Figure 17. PTS Leads PT Timing

PTS is the point of peek angular velocity within a step of the motor. PTS is a function of the slot spacing on the encoder disk, shown in Figure 5. The spacing is determined by the mechanics of the printer mechanism.

When the carriage stepper motor is accelerated from a fixed position, the effects of damping slows the angular velocity of engergizing the stepper motor. This causes PTS to occur after the PT, or PTS lags PT. When PTS lags PT, the next step signal is output at PTS rather than at PT. If the step signal is outputted at PTS, the rotor could be midway through a rotation. Energizing the motor at PT could cause it to bind or shift in the wrong direction. When the carriage stepper motor is at a constant rate of speed, PTS leads PT and the step signal is output at PT (see Figure 13).G. Stored Time Constants.

The time between each step, for a constant number of steps, required for the motor to reach a constant speed, is calculated and stored in Data Memory during acceleration. The values stored are used, in reverse order, during deceleration as the Predetermined Time (PT) Constants. This ensures that the acceleration and deceleration distance traveled by the print head assembly is the same, and that it accurately aligns character dot columns from one line to the next during printing. The time values stored are called "Stored Time Constants." Steps T1 through T11 in Figure 13, represent the Stored Time Constants.

The equations for the Stored Time Constants are given at the bottom of Figure 13, Carriage Stepper Motor Drive Timing.

H. Print Head Assembly "Home" Position

The "logical" Home position for the print head assembly is the left-most position at which printing begins (for L-to-R motion) or ends (for R-to-L motion). The "physical" Home position is the logical HOME position, plus the distance required by the carriage stepper motor to fully accelerate the print head assembly to a constant speed. Printing can only occur when the print head is moving at a constant speed. The printer mechanism manual stipulates eleven step time periods are required to ensure the the print head assembly is at a constant speed. These eleven step time periods are the Stored Time Constants described above. Figure 18 illustrates the components of print head assembly line motion and character printing.

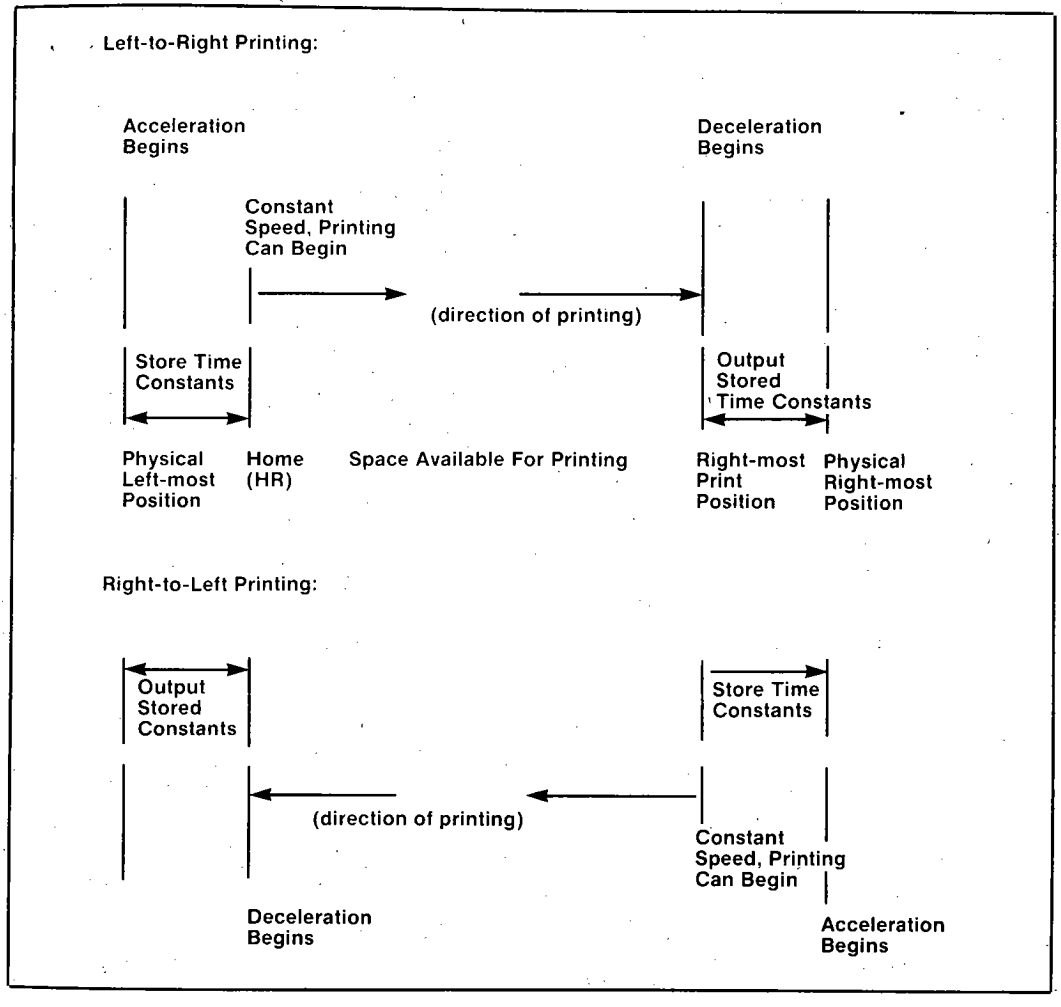


Figure 18. Components of Print Head Assembly Line Motion and Printing

SOFTWARE

Introduction

The software description is presented in three sections. First, a brief overview of the software to familiarize the reader with the interdependencies and overall program flow. Second, data and program memory allocation and status register flag definitions. And third, each of the ten software blocks is presented with its own flowchart.

Software Overview

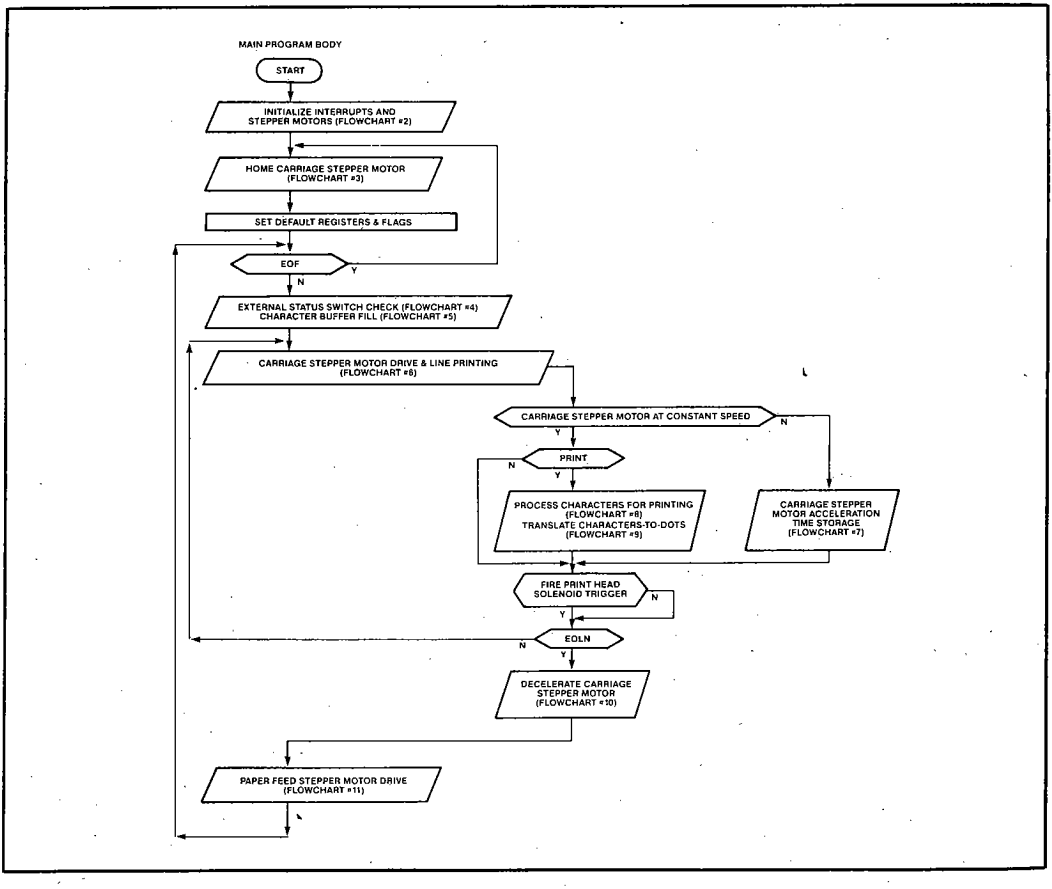
The software is written in Intel UPI-41A/42 Assembly Language. A block structure approach is used for ease of development, maintance, and comprehension. The software is divided into five principal parts.

1. Initialization
2. Character Buffering or Input
3. Stepper Motor Drive and Control
4. Character Processing
5. Character Printing or Output

The five principal parts are incorporated into ten software blocks, listed below.

1. Power On/Reset Initialization
2. Home Print Head Assembly
3. External Status Switch Check
4. Character Buffer Fill
5. Carriage Stepper Motor Drive and Line Printing
6. Accelerate Stepper Motor Time Storage
7. Process Characters for Printing
8. Translate Character-to-Dots
9. Decelerate Carriage Stepper Motor
10. Paperfeed Stepper Motor Drive

Flow Chart No. 1 illustrates the overall software algorithm. Below, is a description of the algorithm.



Flow Chart No. 1. Main Program Body

Upon power-on or reset, a software and hardware initialization is performed. This stablizes and sets inactive the printer hardware and electronics. The print head assembly is then moved to establish its HOME position. The default status registers are set for character buffering, carriage, and paper feed stepper motor drive. The External Status switches are checked; FORMFEED, LINEFEED, ON/OFF LINE, and Character Print TEST. If the printer is ON LINE, the software will loop on filling the Data Memory Character Buffer.

Character or data input to the UPI-42 is interrupt driven. Characters sent by the host system set the Input Buffer Full (IBF) interrupt and the IBF Program Status flag. Character input servicing (completed during the paper feed and carriage stepper motor drive end Delay subroutine) tests for various ASCII character codes, loads characters into the Character Buffer (CB), and repeats until one of several conditions sets the CB Full status flag. Once the CB Full flag is set, further character transmission by the host system is inhibited and printing can begin.

The carriage stepper motor is initialized, and drive begins for the direction indicated. The motor is accelerated to constant speed, printable character codes are translated to dot patterns and printed (if printing is enabled), and the motor is decelerated to a stop. Two timing loops guarantee both constant speed and protection (Failsafe Time) against stepper motor burn out due to high current overload. The two optical sensors, described in the Printer Mechanism section above, are constantly monitored to maintain constant speed, and trigger print head solenoid firing.

Once the line is printed and the carriage stepper motor drive routine has been completed, a Linefeed is forced. The paper feed stepper motor drive subroutine tests the number of lines to move, and energizes the paper feed stepper motor for the required distance. The lines per page default is 66; if 66 lines have been received, a Formfeed to Top-of-Next-Page is performed. The Top-Of-Page is set at Power On/Reset.

When the EOF code is received, the EOF status flag is set. When the last line has been printed, the EOF check will force the print head assembly to the HOME position. The EOF flag is tested following each Paper Feed stepper motor drive. The next entry to the External Status Check subroutine begins a loop which waits for input from either the external status switches or the host system.

The software character dot matrix used in this application is 5 x 7 of the available 9 x 9 print head solenoid matrix. Although lower case descenders and block/line graphics characters are not implemented, Appendix B, Printer Enhancements, discusses how and where these enhancements could be added. The software implements the full 95 ASCII printable characters set.

Memory and Register Allocation

Data Memory Allocation (RAM)

The UPI-42 has 128 bytes of Data Memory. Sixteen bytes are used by the two 8 byte register banks (RB0 and RB1). Sixteen additional bytes are used for the Program Stack. The Stored Time Constants utilize 11 bytes, while the stepper motor phase storage requires 4 bytes. Below is a detailed description of Data and Program Memory

Hex Address	Description

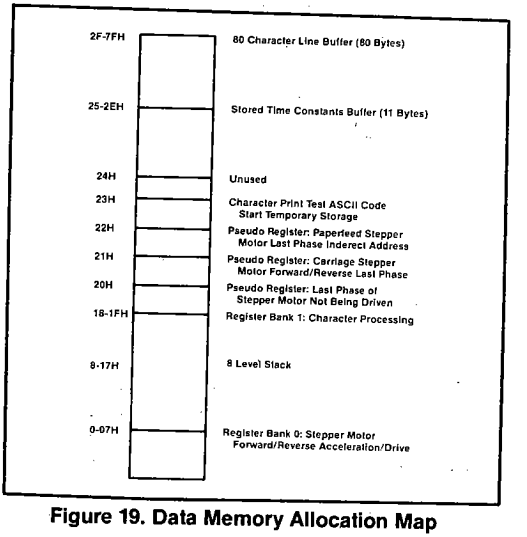


Figure 19. Data Memory Allocation Map

Register Bank 0 is used for stepper motor drive functions. Register Bank 1 is used for character processing. Each register bank's register assignments is listed in Figure 20 and 22, respectively. Each register bank has one register allocated as a Status Register. Figure 21 and 23 detail the Status Register flag assignments. Note that bit 7 of each Status Byte is used as a print head assembly motion direction flag. This saves coding of the Select Register Bank (SEL RBn) instruction at each point the flag is checked.

Register Bank 0

Register	Program Label	Description
R0	TmpR00	RB0 Temporary Register
R1	TStrR0	Store Time Register
R2	GStR20	General Status Register
R3	PhzR30	Stepper Motor Step Register
R4	CntR40	Count Register
R5	TConR0	Time Constant Register
R6	LnCtR0	Line Count Register
R7	OpnR70	Available, Scratch

Figure 20. Register Bank 0 Register Assignment

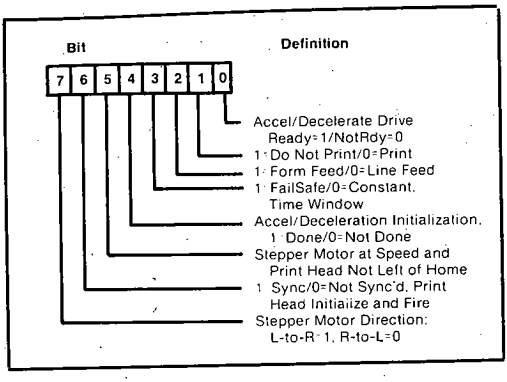


Figure 21. Register Bank 0 Status Byte Flag Assignments

Register	Program Label	Description
R0	TmpR10	RB0 Temporary Register
R1	CAdrR1	Character Data Memory Address Register
R2	ChStR1	Character Processing Status Byte Register
R3	CDtCR1	Character Dot Count Register
R4	CDotR1	Character Dot Temporary Storage Register
R5	CCntR1	Character Count Temporary Register
R6	StrCR1	Store Character Register
R7	OpnR71	Available/Scratch

Figure 22. Register Bank 1 Register Assignment

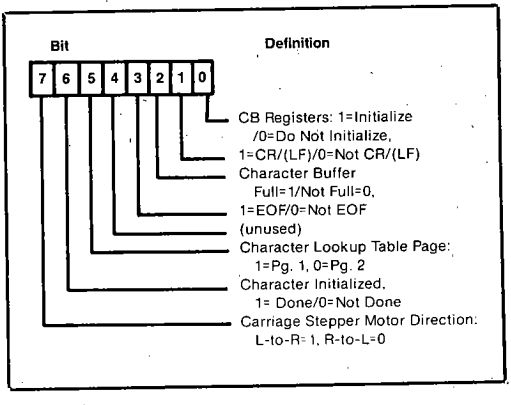


Figure 23. Register Bank 1 Status Byte Flag Assignments

Program Memory Allocation (EPROM/ROM)

The UPI-42 has 2048 bytes of Program Memory divided into eight pages, each 256 bytes. Figure 24 illustrates the Program Memory allocation map by page.

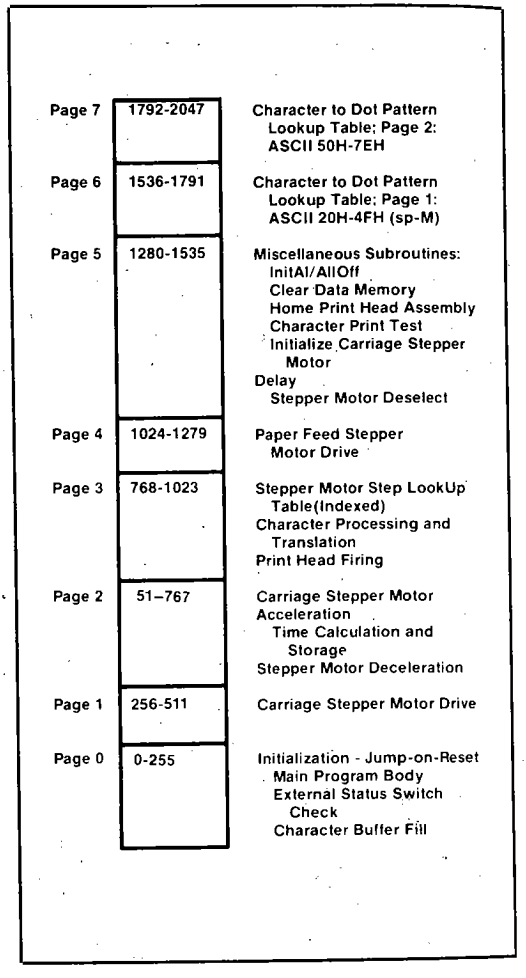


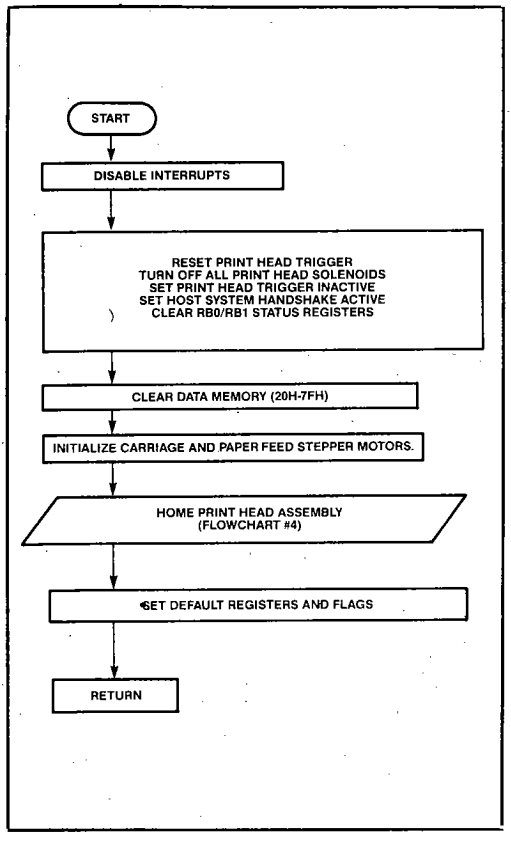
Figure 24. Program Memory Allocation Map

Software Functional Blocks

Below is a decription and flow chart for each of the ten software blocks listed above.

1. Power-On/Reset Initialization

The first operational part in Flow Chart No. 1 is the Power-On or Reset Initialization. Flowchart No. 2 illustrates the Initialization sequence in detail.



Flow Chart No. 2. Power-On/Reset Initialization

Initialization first disables both interrupts. This is done as a precaution to prevent the system software from hanging-up should an interrupt occur before the proper registers and Data Memory values are initialized.

Initialization then deactivates the system electronics. This is also a precaution to protect the printer mechanism and includes the print head solenoid (trigger and data) lines and the stepper motor select lines. The host system handshake signals are activated to inhibit data transfer from the host until the printer is ready to accept data.

Next, Data Memory is cleared from 20H to 7FH. This includes; the 80 byte Character Buffer, the 11 byte Stored Time Constants buffer, and the 4 bytes used as pseudo registers. The pseudo registers are Data Memory locations used as if they were registers. They serve as storage loacations for step data used in accurately reversing the direction of the carriage stepper motor, and stablizing either of the stepper motors not being driven.

The Data Memory locations 00H through 1FH are not cleared. These locations are Register Bank 0 (00H-07H), Program Stack (08H-17H), and Register Bank 1 (18H-1FH) (see Figure 19). Clearing the Program Registers or Stack would cause the initialization subroutine to become lost. The registers are used from the beginning of the program. Care is taken to initialize the registers and stack accurately prior to each program subroutine as required.

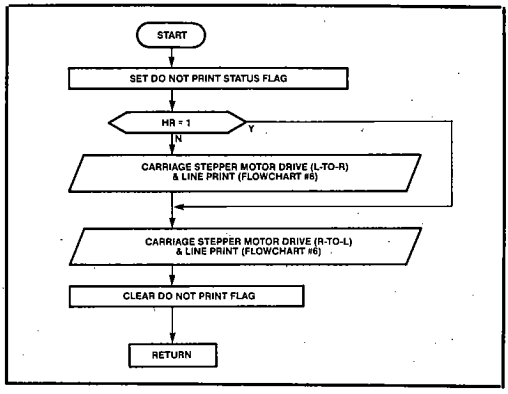
Upon power-on, it is necessary to initialize the two stepper motors, verify their operation, and locate the print head assembly in the left-most 'HOME' position. This sequence serves as a system checkout. If a failure occurs, the motors are deselected and the external status light is turned on. Each stepper motor is selected and energized for a sequence of four steps. This serves to align and stabilize each stepper motor's rotor position, preventing the rotor from skipping or binding when the first drive sequence begins.

At the end of each stepper motor's initialization, the last step data address is stored in one of the Data Memory pseudo registers. The last step data address is recalled at the beginning of the next corresponding stepper motor drive sequence, and used as the basis of the next step sequence. This ensures that the stepper motor always receives the exact next step data, in sequence, to guarantee smooth stepper motor motion. This also garantees the motor never skips or jerks, which would misalign the start, stop, and character dot column positions. A stepper motor not being driven has its last phase data output held constant to stabilize it.

Following any stepper motor drive sequence of either motor, a delay of 30-60 ms occurs by switching the current to Hold Current, stabilizing the motor before it is deselected.

2. Home Print Head Assembly

At the end of the carriage stepper motor four step initialization, the output of the HR optical sensor is tested. The level of the HR signal indicates which drive sequence will be required to 'HOME' the print head assembly. If the print head assembly is to the right of HR, HR is high, the print head assembly need only be moved to from Right-to-Left until HR is low, then decelerated to locate the physical home position. If HR is low, the print head assembly must be moved first Left-to-Right until HR is high, then Right-to-Left to locate both the logical and physical 'HOME' positions. In each case, the software accelerates the carriage stepper motor, generating the Stored Time Constants then decelerates the stepper motor using the Stored Time Constants (see Background section above). Flow Chart No. 3 details the HOME print head assembly subroutine. Figures 13 and 18 illustrate the components of acceleration and print head assembly line motion.



Flow Chart No. 3. HOME Print Head Assembly

The carriage stepper motor drive subroutines used to HOME the print head assembly and to print, are the same. A status flag, called Do-Not-Print, determines whether the Character Processing subroutine is called. The flag is set by the subroutine which calls the Carriage Stepper Motor Drive subroutine. Details of the carriage and paper feed stepper motor drive and character processing subroutines are covered separately below.

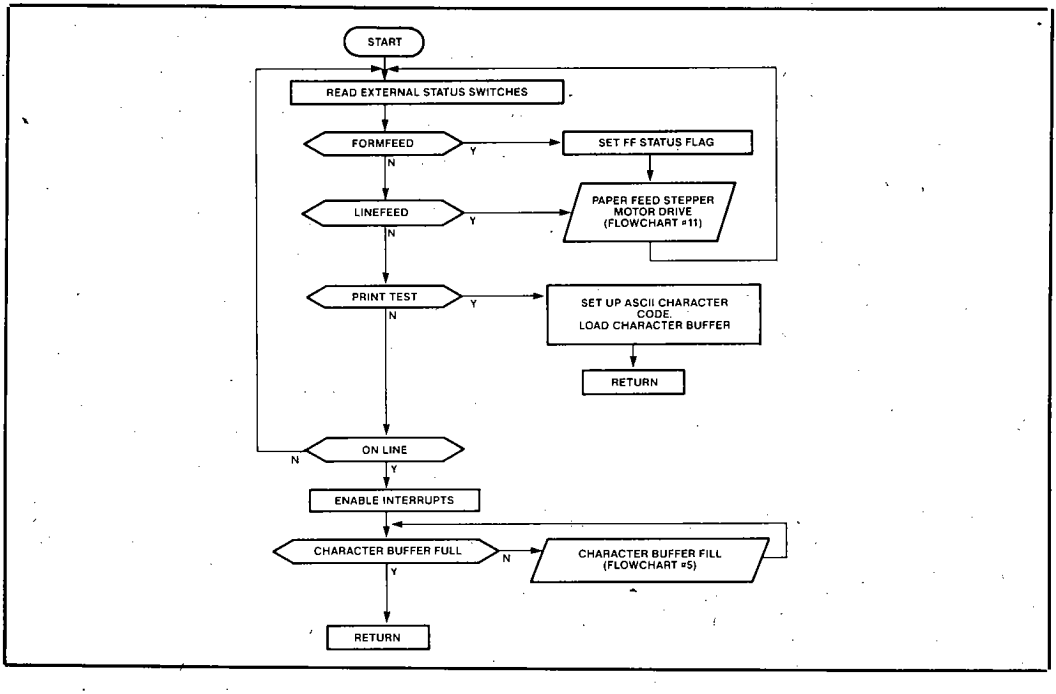
3. External Status Switch Check

Once the system is initialized and the print head is at the HOME position, the software enters a loop which continually monitors the four external status switches, and exits if any one is active. Flow Chart No. 4 details the External Status Switch Check subroutine.

Flow Chart No. 4. External Status Switch Check

If the LINEFEED or FORMFEED switch is set, the Paper Feed subroutine is called. The Paper Feed subroutine is discussed in detail below. If the ONLINE switch is set, the Character Buffer (CB) Fill subroutine is called.

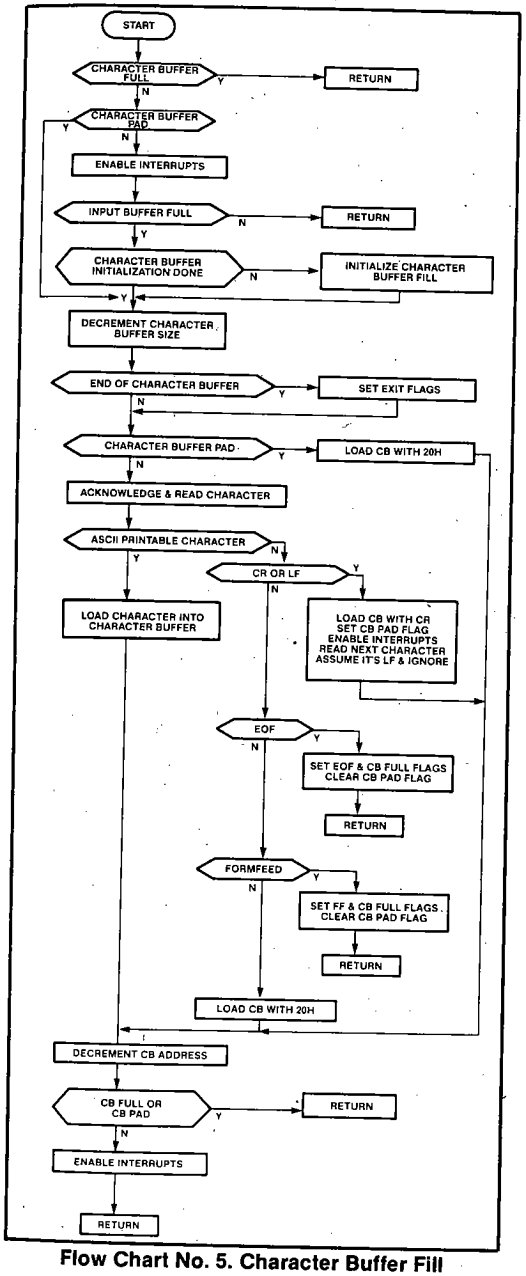
If the Character Print TEST switch is set, the Data Memory Character Buffer (CB) is automatically loaded with the ASCII code sequence, beginning at 20H (a Space character), the first ASCII printable character code. The software then proceeds as if the CB had been filled by characters received from the host system. The External Status Switch Check subroutine is exited and character printing begins. When the line has finished printing, a linefeed occurs (as shown in the main program Flow Chart No.1) and the program returns to the External Status Switch Check subroutine. If the TEST switch remains active, the ASCII character code is incremented and program continues as before. This will eventually print all 95 ASCII printable characters. An example of the TEST printer output, the complete ASCII character code printed, is shown in Figure 25.



Flow Chart No. 4. External Status Switch Check

4. Caracter Buffer Fill

The Character Buffer (CB) Fill subroutine is called from three points within the main program; External Status Switch subroutine, and the Delay subroutine following the carriage and paper feed stepper motor drive subroutines. Flowchart No. 5 details the Character Buffer Fill subroutine operation.



Flow Chart No. 5. Character Buffer Fill

The approximate 80 ms total pre-deselect delay at the end of each stepper motor drive sequence, 40 ms carriage and 40 ms paper feed stepper motor pre-deselect delay, is sufficient to load an entire 80 character line. Half the CB is filled at the end of printing the current line, and the second half is filled at the end of a paper feed. There is no time lost in printing throughput due to filling the character buffer.

Character input is interrupt driven. When the IBF interrupt is enabled, a transmitted character sets the IBF interrupt and IBF Program Status flag. Three instructions make up the IBF interrupt service routine. This short routine disables further interrupts, sets the BUSY handshake line active, inhibiting further transmission by the host, and returns. The subroutine can be executed at virtually any point in the software flow without effecting the printer mechanism operation. Processing of the received character takes place during one of the three program segments mentioned above. The BUSY line remains active until the character is processed by the CB Fill subroutine.

The CB is 80 bytes from the top of Data Memory (30H-7FH). It is a FIFO for forward, left-to-right printing, and a LIFO for reverse, right-to-left, printing. Loading the CB always begins at the top, 7FH. One character may be loaded into the buffer each time the CB Fill subroutine is called.

The CB is always filled with 80 bytes of data prior to printing. If the total number of characters input up to a Carriage Return (CR)/Linefeed (LF), does not completely fill the CB, the CR code is loaded into the CB and the balance of the CB is padded with 20H (Space Character) until the CB is full. A Linefeed (LF) character following a Carriage Return is ignored. A LF is always forced at the end of a printed line. When the CB is full, the CB Full status byte flag is set and printing can begin.

A LF character alone is treated as a CR/LF at the end of a full 80 character line. This is a special case of a printed line and is handled during character processing for printing (see No. 7, Processing Characters for Printing, below). A Formfeed (FF) character sets the FF status byte flag. The flag is tested at each paper feed stepper motor drive subroutine entry.

When the software is available to load the CB with a character, entry to the CB Fill subroutine checks three status flags; CB Full, CB Pad, and IBF flag. If the CB Full flag is set, the program returns without entering the body of the CB Fill subroutine. The CB Pad flag will cause another Space character to be loaded. If the IBF flag is not set, the program returns. If the IBF flag is set, the character is read from the Data Bus Buffer register, tested for printable or nonprintable ASCII code, and, if printable, loaded into the CB. If the character is a non-printable ASCII code and not an acceptable ASCII control code (CR, LF, FF, EOF), a 20H (Space Character) is loaded into the CB.

Exiting the CB Full subroutine with the CB Full or CB

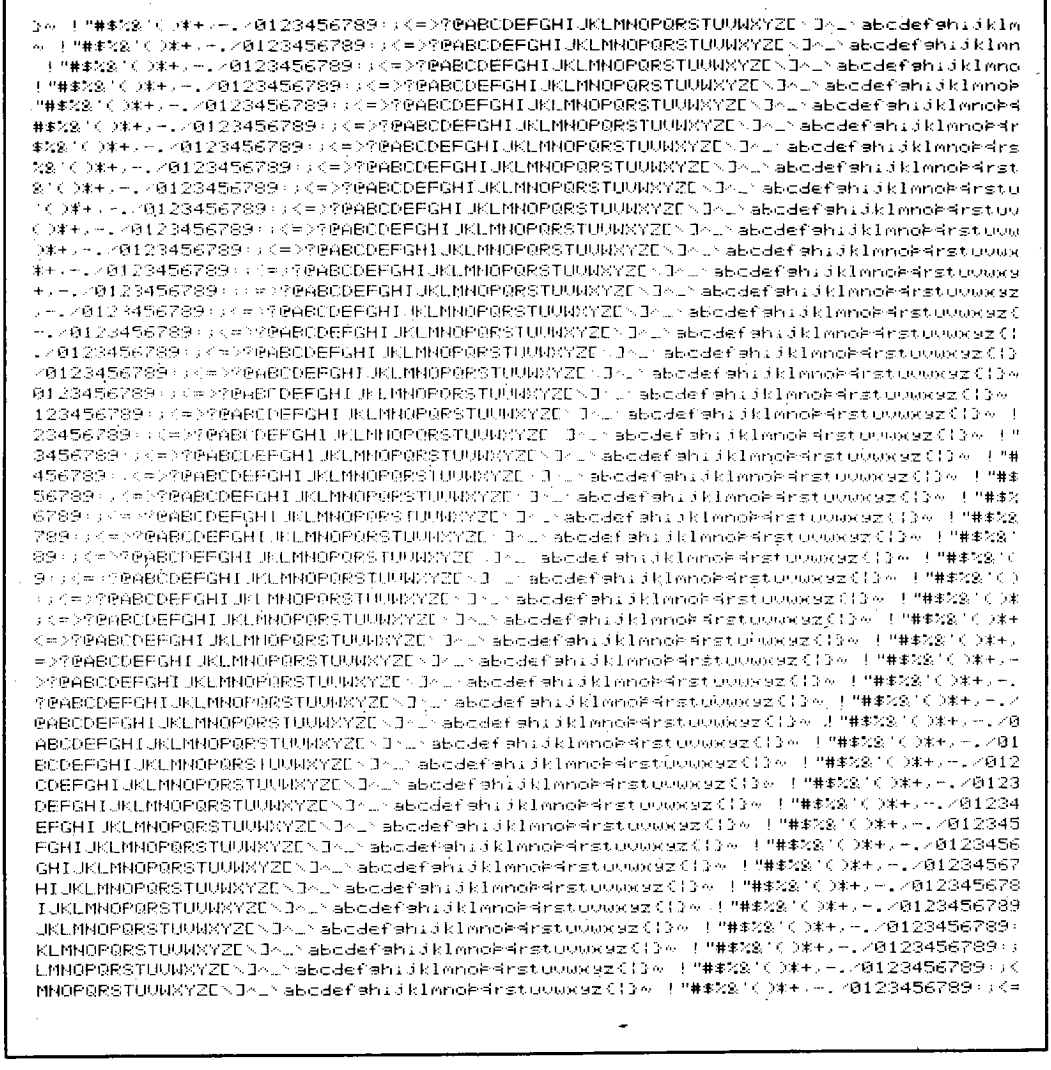


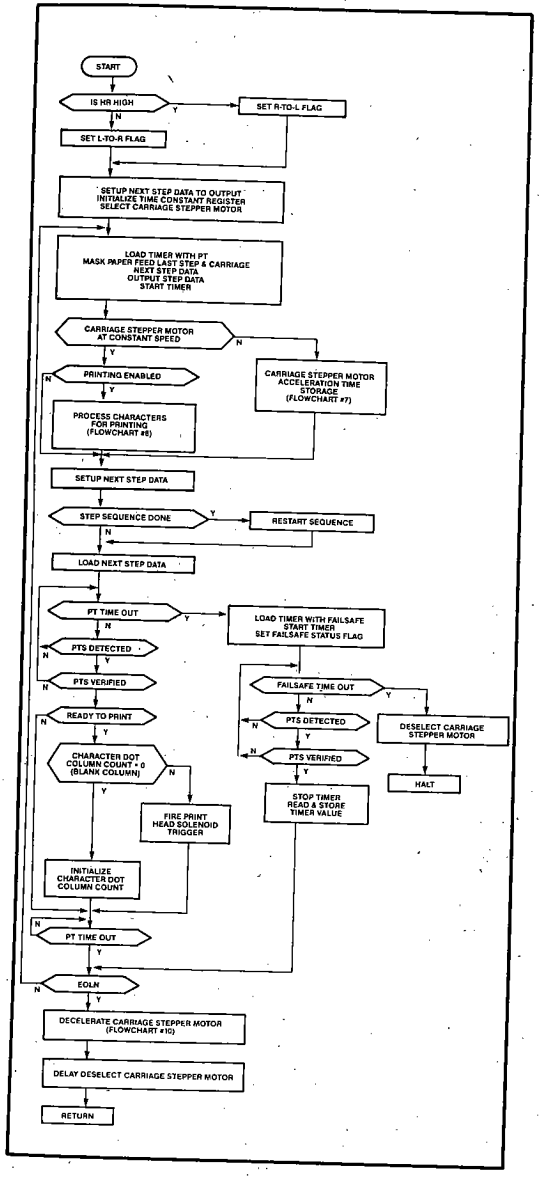
Figure 25. ASCII Character Code TEST Output and Print Example

Pad flag set does not re-enable IBF interrupts or reset the BUSY line. If neither of these flags is set, exiting the CB Fill subroutine sets BUSY inactive and IBF interrupts are enabled. Once the CB Full status byte flag is set, IBF interrupts are disabled until the CB has been entirely emptied, the line printed, or the system Reset.

5. Carriage Stepper Motor Drive and Line Printing

The carriage stepper motor drive subroutine controls both L-to-R and R-to-L print head assembly motion. Upon entering the subroutine, the HR signal level is tested to determine the direction of print head assembly motion and the Direction status flag is set. The default control register values are loaded and balance of the default status flags are set for stepper motor control and character processing. The default control register values include PT and the step sequence look-up table start address for the direction indicated.

The direction flag is tested throughout the carriage stepper motor drive and character processing subroutines. This enables the same subroutines to control activities for either direction, simplifying and shorting the overall program. Flow Chart No. 6 illustrates the carriage stepper motor drive subroutine.



Flow Chart No. 6. Carriage Stepper Motor Drive/Line Printing

Next, the carriage and paper feed stepper motor step data is initialized. The last step data output to the paper feed stepper motor is loaded into the Last Phase pseudo register. This data is masked with each step data output to the carriage stepper motor. Masking the step data in this manner guarantees the paper feed motor signals do not change as the carriage stepper motor is being driven.

Figure 26 illustrates the carriage stepper motor step sequence verses the actual step data output for clockwise rotation, Left-to-Right motion, and counterclockwise rotation, Right-to-Left print head assembly motion. An eight step sequence is depicted in the figure. Note that the sequence for Right-to-Left motion is the reverse of the sequence for Left-to-Right motion. Note also, that for the L-to-R sequence step 4 is the same as step 0, step 5 the same as step 1, etc., through step 7 matching step 3. The four step sequence simply repeats itself until the motor is stopped via the Deceleration subroutine.

L-to-R Motion Sequence	Phase/Step Data (3 2 1 0)	R-to-L Motion Sequence	BCD (3 2 1 0)
0	1 0 0 1	7	0 0 0 0
1	1 0 1 0	6	0 0 0 1
2	0 1 1 0	5	0 0 1 0
3	0 1 0 1	4	0 0 1 1
4	1 0 0 1	3	0 1 0 0
5	1 0 1 0	2	0 1 0 1
6	0 1 1 0	1	0 1 1 0
7	0 1 0 1	0	0 1 1 1

Figure 26. Carriage Stepper Motor Phase/Step Data

When the carriage stepper motor is driven for a specific direction of print head assembly motion, the step sequence must be consistant for the motion to be smooth and accurate. The same holds true for the transition from one direction of motion to the other. Since the sequence for one direction is the opposite for the other direction, incrementing the sequence for L-to-R and decrementing for R-to-L provides the needed step data flow. For example, referring to Figure 26, if the print head assembly moved L-to-R and the last step output was #1, the first step for R-to-L motion would be #7. Thus, when the carriage stepper motor is initialized for a clockwise (L-to-R) or counterclockwise (R-to-L) rotation, the last step sequence number is incremented or decremented to obtain the proper next step. In this way, the smooth motion of the stepper motors is assured.

The step data is referenced indirectly via the step sequence number. The step data is stored in a Program Memory look-up table whose addresses correspond to the step sequence numbers. For example, as shown in

Figure 26, at location 0 the step data "1001" is stored. This method is particularly well suited to the UPI-42 software. The UPI-42 features a number of instructions which perform an indirect move or data handling operation. One of these instructions, MOVP3 A,@A, unlike the others, allows data to be moved from Page 3 of Program Memory to any other page of Program Memory. This instruction allows the step data to be centrally located on Page 3 of Program Memory and accessed by various subroutines.

Each time the carriage stepper motor step data is output, the step data lookup table address is incremented or decremented, depending upon the direction of rotation, and tested for restart of the sequence. The address is tested because the actual step data, Figure 26, is not a linear sequence and thus is not an easily testable condition for restarting the sequence. The sequence number is tested for rollover of the sequence count from 03H to 04H and clockwise motor rotation via the Jump on Accumulator Bit instruction (JBn), with 00H loaded to restart the sequence. The same bit is tested when decrementing the sequence count for counterclockwise motor rotation, R-to-L motion, because the count rolls over from 00H to 0FFH, with 03H loaded to restart the sequence.

At this point the UPI-42 Timer/Counter is loaded, the step signal is output, and the timer started. The next step data to be output has been determined and the At-Speed flag is tested for entry to one of two subroutines; Stepper Motor Acceleration Time Storage or Character Processing.

The first entry to the Acceleration Time Storage subroutine initializes the subroutine and returns. All other entries to one of the two subroutines perform the necessary operations, detailed below (Blocks 6 and 7), and returns. The program loops until the PT times out or the PTS level change is detected. PTS is tied to T0 of the UPI-42. The level present on T0 is directly tested via conditional jump instrunctions. The software loops on polling the timer Time Out Program Status flag and the T0 input level.

As described in the Background section above (shown in Figure 13), if PT times out before PTS is detected, the software waits for PTS before outputing the next step signal. If PT times out before PTS, a second timer count value is loaded into the UPI-42 timer. The timer value is called "Failsafe." This is the maximum time the stepper motor can be selected, with no rotor motion, and not damage the motor. If PTS is not detected, either the carriage stepper motor is not rotating or the optical sensor is defective. In either case, program excution halts, the motor is deselected, and the external status light is turned on to indicate a malfunction. A system reset is required to recover from this condition. The Failsafe time is approximately 20 milliseconds, including PT.

The Failsafe time loop also serves as a means of tracking the elapsed time between PT time out and PTS. Entry to the Failsafe time loop sets the Failsafe/Constant Time Window status flag. This flag is tested by the Acceleration Time Storage subroutine for branching to the proper time storage calculation to be perform (see Figure 13 and Block 6 below for further description).

During the Failsafe timer loop, if PTS is detected and verified as true, the Failsafe timer value is read and stored in the Time Storage register. This value is used during the next Acceleration Time Storage subroutine call to calculate the Stored Time Constant (see Block 6 below). If PTS is invalid, the flow returns to the timer loop just exited, again waiting for PTS or Failsafe time out.

During the PT time loop, if PTS is detected and verified, the Sync flag is tested for entry to the print head solenoid firing subroutine. This flag is set by the first entry to the Character Processing subroutine. The flag synchronizes the solenoid firing with character processing. Only if characters are processed for printing will the solenoids be enabled, via the Snyc flag, for firing. This prevents the solenoids from being fired without valid character dot data present.

As described in the Background section "Relationship Between PTS and PT," PTS is the point of peek angular velocity within a step of the motor. After PTS is detected the motor speed ramps down, compensating for the overshoot of the rotor motion. PTS is the optimum time for print head solenoid firing, as shown in Figure 13. This is the most stable point of motor rotation and, thus, the print head assembly motion. If PTS is detected during PT, printing is enabled, the Sync flag is set, and the solenoid trigger is fired.

The firing of the solenoid trigger, following PTS, is very time critical. The time between PTS and solenoid firing must be consistant for accurate dot column alignment throughout the printed line. The software is designed to meet this requirement by placing all character processing and motor control overhead before the solenoid firing subroutine is called. The actual instruction sequence which fires the print head solenoid trigger is plus or minus one instruction for any call to the subroutine.

Once the timer loop is complete, the software tests for Exit conditions. If the Exit conditions fail, the software loops to output the next step signal, starts the PT timer, and continues to accelerate the carriage stepper motor, or process, and print characters. If the Exit test is true, the carriage stepper motor is decelerated to a fixed position, and the program returns to the main program flow (see Flowchart 1).

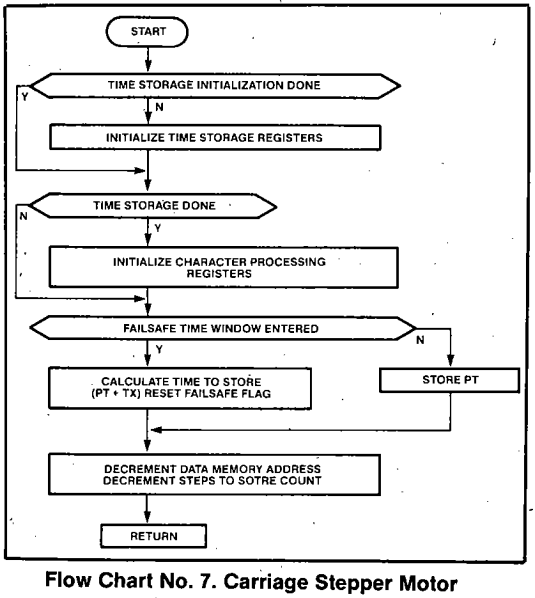
The exit conditions are different for the two directions of print head assembly motion. For L-to-R printing, if a Carriage Return (CR) character code is read from CB, the carriage stepper motor drive terminates and the motor is decelerated to a fixed position. There are two conditions for terminating carriage stepper motor drive upon detecting a CR during L-to-R motion. If less than half a character line (40 characters) has been printed,

the print head assembly returns to the HOME position to start the next printed line. Otherwise, the print head assembly continues to the right-most position for a full 80 character line, and then begins printing the next line from R-to-L. R-to-L printing always returns the print head assembly to the HOME position before the next line is printed L-to-R. When HR is high, character printing always stops and the carriage stepper motor drive subroutine exits to the deceleration subroutine.

6. Accelerate Stepper Motor Time Storage

As described above, when the carriage stepper motor is accelerated the step time required to guarantee the motor is at a constant rate of speed translates to a specific distance traveled by the print head assembly (see Figure 18). In order to position the print head assembly accurately for bi-directional printing, the distance traveled during deceleration must be the same as during acceleration. The Carriage Motor Acceleration Time Storage subroutine calculates the step times needed to accelerate the carriage stepper motor, and stores them in Data Memory for use as PT during deceleration.

The first call of the Carriage Stepper Motor Acceleration Time Storage subroutine initializes the required registers and status flags. The time calculation begins with the second carriage stepper motor step signal output. The program returns to the carriage stepper motor drive subroutine and loops on PT. Each subsequent call of the Acceleration Time Storage subroutine tests the Failsafe/Constant flag and branches accordingly (see Flow Chart 7). The Acceleration Time Storage subroutine has two parts which correspond to PTS leading or PTS lagging PT.



Flow Chart No. 7. Carriage Stepper Motor Acceleration Time Storage

If the Failsafe/Constant flag is set, PTS lagged PT. The time from PT time out to PTS, Tx (see Figure 13), must be added to the PT and stored in Data Memory. As described above, if PT lagged PT, the Failsafe time is loaded and PTS is again polled during the time loop. When PTS occurs within the Failsafe time, the timer is stopped and the timer value stored. The UPI-42 timer is an up timer, which means that the value stored is the time remaining of the Failsafe time when PTS occured. The elapsed time must be calculated by subtracting the time remaining (the value stored) from the Failsafe time constant. This is done in software by using two's complement arithmetic. If the Failsafe flag is not set PTS led PT, and PT is the Stored Time Constant stored.

Indirect addressing of Data Memory is used to reference the Stored Time Constant Data Memory location. The Data Memory location address is decremented each time the Acceleration Time Storage subroutine is exited and a Stored Time Constant has been generated.

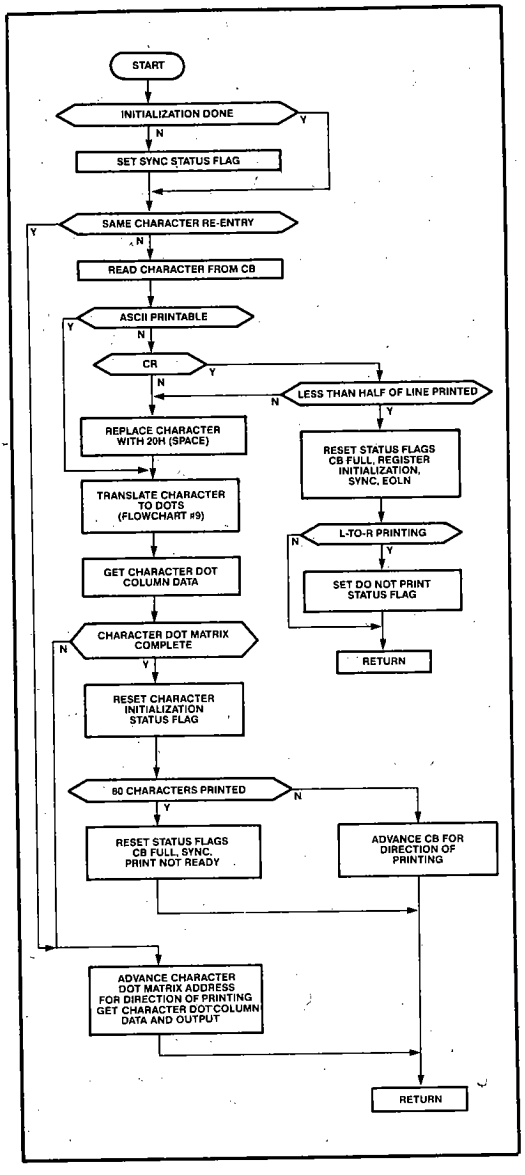
The last Acceleration Time Storage subroutine exit sets the At-Speed status flag and initializes the character processing registers and flags.

3. Process Characters for Printing

The Character Processing subroutine is entered only if the Home Reset (HR) optical sensor signal is high and printing is enabled. Otherwise, the software simply returns to the Carriage Stepper Motor Drive subroutine. There are two cases when printing is not enabled; during the HOME subroutine operation, and when the print head assembly returns to the HOME position after printing less than half an 80 character line. If printing is enabled, the Sync status flag is set.

All character processing operations use the second UPI-42 Data Memory Register Bank, RB1. Register Bank 1 is independent of Data Memory Register Bank 0, used for stepper motor control. The use of two independent register banks greatly simplfies the software flow, and helps to ensure the accuracy of event sequences that must be handled in parallel. Each register bank must be initialized only once for any entry to either the Carriage Stepper Motor Drive or Character Processing subroutines. A single UPI-42 Assembly Language instruction selects the appropriate register bank. Initializing the character processing registers includes loading the maximum character count (80), dot matrix size count (6), and CB start address. The CB start address is print direction dependant, as described in Block 4, above.

Character processing reads a character from the CB, tests for control codes, translates the character to dots, and conditionally exits, returning to the Carriage Stepper Motor Drive subroutine. Flow Chart 8 details the character processing subroutine.



Flow Chart No. 8. Process Characters for Printing

Each character requires six steps of the carriage stepper motor to print; five for the 5 character dot columns and 1 for the blank dot column between each character. Reading a character from the CB and character-to-dot pattern translation takes place during the last character dot column, or blank column, time.

The first character line entry to the Character Processing subroutine appears to the software as if a last character dot column (blank column) had been entered. The next character, in this case the first character in the line, is translated and printing can begin. This method of intiializing the Character Processing subroutine utilizes the same software for both start-up and normal character flow. Once a character code has been translated to a dot matrix pattern starting address in the look-up table, all subsequent entries to the Character Processing subroutine simply advance the dot column data address and outputs the data.

The decision to translate the character to dots during the blank column time was an arbitary one. As was the choice of the blank column following rather than preceding the actual character dot matrix printing.

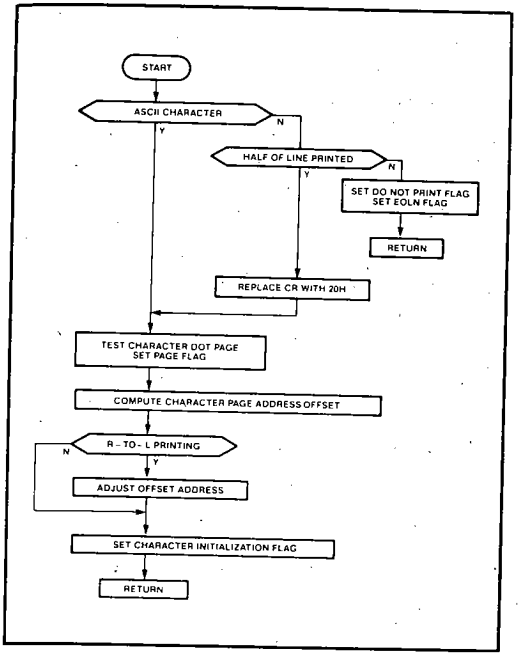
4. Translate Character-to-Dots

Character-to-dot pattern translation involves converting the ASCII code into a look-up table address, where the first of the five bytes of charcter dot column data is stored. The address is then incremented for the next column of dot pattern data until the full character has been printed.

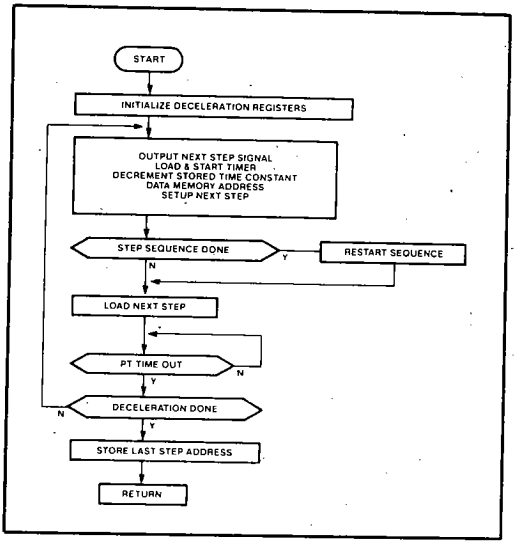
The dot pattern look-up table occupies two pages, or approximately 512 bytes of Program Memory. A printable ASCII character is tested for its dot pattern location page and the offset address, from zero, on that page. Both the page test and page offset calculations use two's complement arithmetic, with a jump on carry or not carry causing the appropriate branching. Once the pattern page and address are determined the indirect addressing and data move instructions are used to read and output the data to the print head solenoids. Flowchart 9 details the Character-to-Dots Translation subroutine.

In the case of R-to-L printing, although the translation operation is the same, the character is printed in reverse. This requires that the character dot pattern address be incremented by five, before printing begins, so that the first dot column data output is the last dot column data of the character. The dot pattern look-up table address is then decremented rather that incremented, as in L-to-R printing, for the balance of the character. Translation still takes place during the last character dot column, the blank column, and the blank column follows the character matrix.

Only one control code, a Carriage Return (CR), is encountered by the character translation subroutine. Linefeed (LF) characters are stripped off by the CB Fill subroutine. If a CR code is detected the software tests for a mid-line exit condition; less than half the line printed exits the stepper motor drive subroutine and HOMEs the print head assembly before printing the next line. If the test fails, more than half the line has been printed, the CR is replaced by a 20H (Space character) and printing continues for the balance of the line; the space characters padding the CB are printed.



Flow Chart No. 9. Translate Character-to-Dots



Flow Chart No. 10. Decelerate Carriage Stepper Motor

As mentioned above, the character dots are printed and the print head trigger is fired when the PTS signal is detected and verified and the carriage stepper motor is At Speed.

When the character to print test fails the CB Buffer size count equals zero, the Carriage Stepper Motor Drive subroutine exit flags are set, and the flow passes to the Deceleration and Delay subroutines and programs returns to the main program flow.

9. Decelerate Carriage Stepper Motor

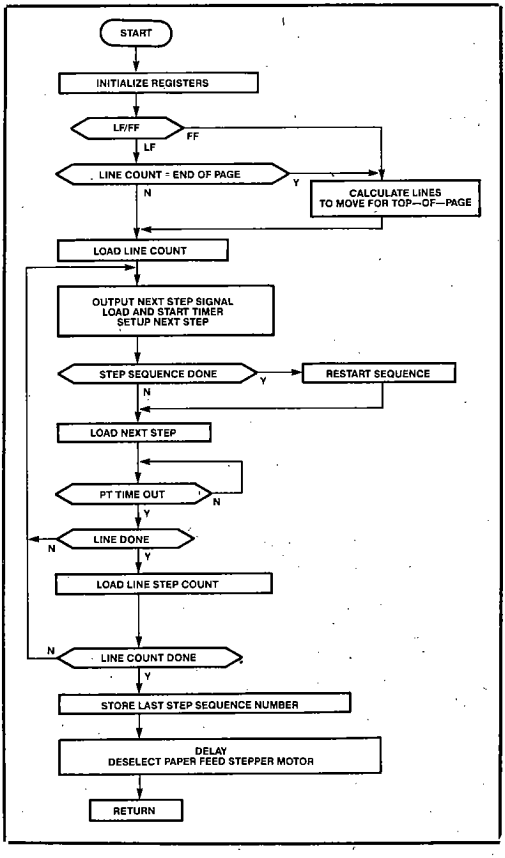
The transition from the Carriage Stepper Motor Drive subroutine to the Deceleration subroutine outputs the next step signal in sequence, and then initializes the Deceleration subroutine registers; Stored Time Constants Data Memory buffer end address and size. The Stored Time Constant Buffer is a LIFO for deceleration of the carriage stepper motor. The buffer size is used as the step count. When the step count decrements to zero, the step signal output is terminated, and the last step sequence number is stored in the carriage stepper motor Next Step pseudo register. The last step sequence number is recalled, during initialization of the next carriage stepper motor drive, as the basis of the next step data signal to be output. See Flow Chart 10.

When the carriage stepper motor is decelerated, Failsafe protection and PTS monitoring are not necessary. The Deceleration subroutine acts as its own failsafe mechanism. Should the stepper motor hang-up, the subroutine would exit and deselect the motor in sufficient time to protect the motor from burnout. Since neither Failsafe nor print head solenoid firing take place during deceleration, PTS is not needed. PT is replaced by the Stored Time Constant values in Data Memory. The Deceleration subroutine determines the next step signal to output, loads the Timer with the Stored Time Constant, starts the UPI-42 Timer, and loops until time out. The subroutine loops to output the next step until all of the Stored Time Constants have been used. The program returns to the Carriage Stepper Motor Drive subroutine and the motor is deselected following the Delay subroutine execution. The Delay subroutine is called to stablize the stepper motor before it is deselected. During the DELAY subroutine, the IBF interrupt is enabled and characters are processed. A paper feed is forced following the carriage stepper motor being deselected.

10. Paper Feed Stepper Motor Drive

The paper feed stepper motor subroutine outputs a predefined number of step signals to advance the paper, in one line increments, for the required number of lines. The number of step signals per line increment is a function of the defined number of lines per inch, given the distance the paper moves in one step. Figure 16 lists three step (or pulse) count and line spacing configura-

tions, as well as the distance the paper moves in one step. Standard 6 lines per inch spacing has been implemented in this Application Note (Appendix B details how variable line spacing could be implemented). Flowchart 11 illustrates the Paper Feed subroutine.



Flow Chart No. 11. Paper Feed Stepper Motor Drive

The number of lines the paper is to be moved is called the "Line Count." The Line Count defaults to one unless the Formfeed flag is set, or the total number of lines previously moved equals a full page. The default total lines per page for this application is 66. When the total number of lines moved equals 66, the paper is moved to the top of the next page. The Top-of-Page is set at power-on or reset.

If the Formfeed flag has been set in the Character Buffer Fill subroutine, the software calculates the number of lines needed for a top of next page paper feed. The resulting line count is loaded in the Line Count Register. The Paper Feed subroutine loops on the line count until done and then returns to the main program body.

Once the Paper Feed subroutine is complete, the software loops to test the End of File (EOF) Flag (see Flow Chart 1). If EOF is set, the print head assembly is moved to the HOME position, the program again enters the External Status Switch Test subroutine, and begins polling the external status switches. If EOF is not set, the program directly calls the External Status Switch Check subroutine, and the program repeats for the next line.

CONCLUSION

Although the full speed, 12 MHz, of the UPI-42 was used, the actual speed required is approximately 8-9 MHz. 1400 bytes of the available 2K bytes of Program Memory were used; 500 bytes for the 95 character ASCII code dot pattern look-up table, 900 bytes for operational software. This means that the UPI-42 has excess processing power and memory space for implementing the additional functions such as those listed below and discussed in Appendix B.

Special Characters or Symbols
Lower Case Descenders
Inline Control Codes
Different Character Formats
Variable Line Spacing

The software developed for this Application Note was not fully optimized and could be further packed by combining functions. This would require creating another status register, which could also serve to implement some of the features listed above. Since the full 16 byte stack is not used for subroutine nesting, there are 6-8 bytes of Program Stack Data Memory that could be used for this purpose. In several places, extra code was added for clarity of the Application Note. For example, each status byte flag is set with a separate instruction, using a equate label, rather than setting several flags simultaneously at the same point in the code.

This Application Note has demonstrated that the UPI-42 is easily capable of independently controlling a complex peripheral device requiring real time event monitoring. The moderate size of the program required to implement this application attests to the effectiveness of the UPI-42 for peripheral control.

APPENDIX A.
SOFTWARE LISTING

```
   1 $MOD42  TITLE('UPI 42 APP NOTE');
   2 $MACROFILE NOSYMBOLS NOGEN DEBUG
   3
   4 $INCLUDE(:F1:ANECD.OV1)
=  5 ; PG
=  6
=  7 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
=  8 ;               Complex Peripheral Control With the UPI-42
=  9
= 10 ;               Intel Corporation
= 11 ;               3065 Bowers Avenue
= 12 ;               Santa Clara, Ca. 95051
= 13
= 14
= 15 ;               Written By      Christopher Scott
= 16
= 17
= 18 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 19
= 20 ;               Notes and Comments
= 21
= 22 ;       Three Assembly Language files comprise the full Application
= 23 ;       Note source code;
= 24
= 25 ;       1. ANECD.OV1     App Note Equates, Constants, Declarations . Overlay
= 26
= 27 ;       2. 42ANC.SRC     UPI-42 App Note Code Source
= 28
= 29 ;       3. CHRTBL.OV1    Character Table . Overlay  (Character Lookup Tables)
= 30
= 31
= 32
= 33
= 34 ; PG
= 35 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 36 ;               Equates, Constants and System Definitions
= 37 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 38
= 39 ;       Data & Program Memory Allocations
= 40
= 41 ;       Program Memory
= 42
= 43 ;       Page No.   Hex Addr      Description
= 44 ; ---------------------------------------------------------------------------
= 45
= 46 ;       Page 7   1792-2047      Char to Dot pattern lookup table
= 47 ;                                 Page 2:  ASCII 50H-7FH (N-~)
= 48 ;       Page 6   1536-1791      Char to Dot pattern lookup table
= 49 ;                                 Page 1:  ASCII 20H-4FH (sp-M)
= 50 ;       Page 5   1280-1535      Misc called routines:
= 51 ;                                , InitAl/AllOff
= 52 ;                                 Clear Data Memory
= 53 ;                                 CR Home
= 54 ;                                 Char Print Test -  load Ascii char codes
= 55 ;                                 Initialize CR Stpr Mtr
= 56 ;                                 Delay: short/long/very long
= 57 ;                                 Stpr Mtr deselect
= 58 ;       Page 4   1024-1279      PaperFeed Stpr Mtr Init and Drive
= 59 ;       Page 3    768-1023      Stpr Mtr Phase LookUp Table - Indexed
= 60 ;                                Character Translation and processing
= 61 ;                                PrintHead firing
= 62 ;       Page 2    512-767       Stpr Mtr Accel. Time calc. and memorization
= 63 ;                                Stpr Mtr Deceleration
= 64 ;       Page 1    256-511       SMDriv (FAccel/RAccel) - Forward & Reverse
= 65 ;                                 Stpr Mtr acceleration & drive
= 66 ;       Page 0      0-255       Initiaization Jmp-on-Reset
= 67 ;                                Program Body - all calls
= 68 ;                                Character Input test and Char Buffer fill loop
= 69 ;                                Interrupt service routines
= 70
```

```
= 71 ; PG
= 72 ; ------------------------------------------------------------------------------
= 73 ;       Data Memory
= 74 ; ------------------------------------------------------------------------------
= 75 ;         Dec.        Hex          Description
= 76 ;        -------     --------      ------------------------------------------
= 77 ;    TOP
= 78 ;    ---------------------
= 79 ;       48-127      2F-7FH        80 Character Line Buffer
= 80 ;        37-47      25-2EH        Stpr Mtr Accel/Decel time, memorization
= 81 ;           36         24H        Unused
= 82 ;           35         23H        Char Print test ASCII code start tmp store
= 83 ;           34         22H        LF SM last Phz Inderect Addr psuedo reg
= 84 ;           33         21H        CR SM Forward/Reverse last Phz psuedo reg
= 85 ;           32         20H        Psuedo Reg:  Last Phase of stpr mtr not
= 86 ;                                                 being driven
= 87 ;        24-31      18-1FH        Register Bank 1:  Character Handling
= 88 ;         8-23       8-17H        8 Level Stack
= 89 ;         0-7        0-07H        Register Bank 0:  Stpr Mtr F/R Accel/Drive
= 90 ;    ---------------------
= 91 ;    BOTTOM
= 92
= 93 ;         Data Memory Equates:
= 94 ; ------------------------------------------------------------------------------
= 95
0050  = 96 CHBFSZ  EQU     50H             ;char buffer size 0-79 = 80
00D9  = 97 H1fCpl  Equ     0D9H            ;Cpl(1/2 CbBfSz) => cpl of 27H = 0D9H
      = 98
007F  = 99 FCBfSt  Equ     7fH             ;start of char buffer
0080  = 100 FCBfIS  Equ    80H             ;init CB strt-allows xtra Dec by 1
002F  = 101 RCBfIS  Equ    2FH             ;init CB strt-allows xtra Inc by 1
0051  = 102 ChBfIS  Equ    81              ;load char cnt reg w/char bufr Init Size
      = 103
002F  = 104 ENDBUF  EQU    2FH             ;END OF CHAR BUFFER
0008  = 105 ASBfSz  EQU    0BH             ;Accelerate stpr mtr buf count
000A  = 106 DSBfSz  Equ    0AH             ;Decelerate stpr mtr buf count
002F  = 107 SMBFST  EQU    2FH             ;STPR MTR BUFFER START
0025  = 108 SMBEnd  Equ    25H             ;Stpr Mtr Data Memory Address end
007F  = 109 DMTop   Equ    7FH             ;Data Memory Top
005D  = 110 DMSize  Equ    93              ;Data Memory Size (less two working reg's)
      = 111
0020  = 112 LastPh  Equ    20H             ;last phz psuedo reg addr
0021  = 113 CPSAdr  EQU    21H             ;CR phz psuedo reg
0022  = 114 LPSAdr  Equ    22H             ;LF phz psuedo reg
0023  = 115 PTAscS  Equ    23H             ;Char Print Test code start tmp store
      = 116
= 117 ; PG
= 118 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 119 ;       Register allocation
= 120 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 121
= 122 ;         All Indirect Data Memory Addressing via @Rn Inst must use
= 123 ;           only registers 0 & 1 of either register bank.  Any other will
= 124 ;           be rejected by the Assembler
= 125 ;         Last character in lable indicates Register Bank referenced
= 126
= 127 ;       Register Bank 0
= 128 ; ------------------------------------------------------------------------------
0000  = 129 TmpR00  Equ    R0              ;RB0 Temporary Register
0001  = 130 TStrR0  EQU    R1              ;Store Time Register RB0
0002  = 131 GStR20  EQU    R2              ;General Status Register RB0
0003  = 132 PhzR30  EQU    R3              ;Stpr Mtr Phase Register RB0
0004  = 133 CntR40  Equ    R4              ;Count Reg. Phase count-Stpr Mtr loops
      = 134                               ;  Accel/Decel Count
0005  = 135 TConR0  Equ    R5              ;Time constant reg RB0
0006  = 136 LnCtR0  Equ    R6              ;Line count
      = 137
0007  = 138 OpnR70  EQU    R7              ;available
      = 139
= 140 ;         Register Bank 0 Data Memory Address
= 141 ; ------------------------------------------------------------------------------
```

```
0000            = 142 TmpA00  Equ     00H             ;Temporary Register DM address
0001            = 143 TStrA0  EQU     01H             ;Time Store Register  DM address
0002            = 144 GStRAd  Equ     02H             ;RB0 Char Status Reg  DM address
0003            = 145 PhzA20  EQU     03H             ;Stpr Mtr Phase Register DM address
0004            = 146 CntRA0  Equ     04H             ;Count Reg. Phase count-Stpr Mtr loops
                = 147                                 ;  Accel/Decel Count DM address
0005            = 148 TConA0  Equ     05H             ;Time constant reg DM address
0006            = 149 LnCtA0  Equ     06H             ;Line Count Register DM address
                = 150
0007            = 151 OpnA70  EQU     07H             ;available
                = 152
                = 153
                = 154 ; PG
                = 155 ; ------------------------------------------------------------------------
                = 156 ;       RB0  Status Byte Bit Definition
                = 157 ; ------------------------------------------------------------------------
                = 158
                = 159 ;       Bit             Definition
                = 160 ;       -----           ----------------------------------------------------
                = 161 ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
                = 162 ;       6               1 = Sink / 0 = Not Sinked, Print Head Init and Fire
                = 163 ;       5               Stpr Mtr at speed and CR not left of Home
                = 164 ;       4               Accel/Decel Init, 1 = Done / 0 = Not Done
                = 165
                = 166 ;       3               1 = FailSafe / 0 = Constant, Time Window
                = 167 ;       2               1 = Form Feed / 0 = Line Feed
                = 168 ;       1               1 = Do Not Print / 0 = Print
                = 169 ;       0               FAccel/DAccel drive Ready = 1/NotRdy = 0 (exit
                = 170 ;                         drive & decel stpr mtr)
                = 171
                = 172 ;       Bit Masks:      RB0
                = 173 ;       Stepper Motor control bit masks function on GStR10
                = 174 ; ------------------------------------------------------------------
                = 175
                = 176 LRPrnt  Equ     80H             ;Left to Right Printing (ORL)
                = 177 RLPrnt  Equ     7FH             ;Right to Left Printing (ANL)
                = 178 SnkSet  Equ     40H             ;Ready Print flag
                = 179 ClrSnk  Equ     0BFH            ;clear Ready to Print Bit
                = 180 AtSpdF  Equ     20H             ;Stpr Mtr at constant speed
                = 181 NAtSpd  Equ     0DFH            ;Stpr Mtr Not at speed
                = 182 ADIntD  Equ     10H             ;Accel/Decel Init Done
                = 183 ADIntN  Equ     0EFH            ;Accel/Decel Init Not Done
                = 184
                = 185 FsCTm   Equ     08H             ;FailSafe/Constant Time
                = 186 ClrFsC  Equ     0F7H            ;Clear FailSafe/Const time flag
                = 187 FrmFd   Equ     04H             ;do formfeed
                = 188 LineFd  Equ     0FBH            ;do line feed
                = 189 DoNotP  Equ     02H             ;set Do Not Print Stat bit
                = 190 OkPrnt  Equ     0FDH            ;Reset - Ok to Print
                = 191 Ready   Equ     01H             ;Ready drive Stpr Mtr
                = 192 NotRdy  Equ     0FEH            ;Not Ready exit Stpr Mtr drive
                = 193
                = 194
                = 195 ; PG
                = 196 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 197 ;       Register allocation   (cont)
                = 198 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 199
                = 200
                = 201 ;       Register Bank 1
                = 202 ; ------------------------------------------------------------------
0000            = 203 TmpR10  Equ     R0
0001            = 204 CAdrR1  EQU     R1              ;char data memory addr register
0002            = 205 ChStR1  EQU     R2              ;char processing status byte register
0003            = 206 CDtCR1  EQU     R3              ;Char Dot count register
0004            = 207 CDotR1  Equ     R4              ;Char dot temp storage register
0005            = 208 CCntR1  Equ     R5              ;Char count temp register
0006            = 209 StrCR1  EQU     R6              ;Store Char Register
                = 210
0007            = 211 OpnR71  EQU     R7              ;Available
                = 212
                = 213 ;       Register Bank 1 Data Memory Address
                = 214 ; ------------------------------------------------------------------
```

```
0018          = 215 TmpA1O  Equ     24              ;temporary/scratch register
0019          = 216 ChARR1  EQU     25              ;char data memory addr register
001A          = 217 ChStAd  Equ     26              ;RB1 Char Status Reg address
001B          = 218 CDtCA1  EQU     27              ;Char Dot count register
001C          = 219 CDotA1  Equ     28              ;Char dot temp storage register
001D          = 220 CCntA1  Equ     29              ;Char count temp register
001E          = 221 StrCA1  EQU     30              ;Store Char Register
              = 222
001F          = 223 OpnA71  EQU     31              ;Available
              = 224
              = 225
              = 226 ; PG
              = 227 ; ------------------------------------------------------------------------
              = 228 ;       RB1  Status Byte Bit Definition
              = 229 ; ------------------------------------------------------------------------
              = 230
              = 231 ;       Bit             Definition
              = 232 ;      -----           ------------------------------------------
              = 233 ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
              = 234 ;       6               Char Init, 1 =  Done / 0 = Not Done
              = 235 ;       5               Char Lookup Table Page: 1 = Pg1, 0 = Pg2
              = 236 ;       4               1 = Test / 0 = Normal char print/input
              = 237
              = 238 ;       3               1 = EOF / 0 = Not EOF
              = 239 ;       2               Full = 1/Not Full = 0, Line in Char Buffer
              = 240 ;       1               1 = CR/(LF) / 0 = Not CR/(LF)
              = 241 ;       0               1 = Init / 0 = Do Not Init, CB registers done
              = 242
              = 243 ;       Bit Masks:      RB1
              = 244 ;       Character printing bit masks function on ChStR1
              = 245 ; ------------------------------------------------------------
0080          = 246 ChrPrn  Equ     80H             ;Stpr Mtr Direction:  L-to-R = 1
007F          = 247 ClrCPr  Equ     7FH             ;Stpr Mtr Direction:  R-to-L = 0
0040          = 248 ChIntD  Equ     040H            ;Set Char Init Done
00BF          = 249 CIntND  Equ     0BFH            ;Reset Char Init Not Done
0020          = 250 ChOnP1  Equ     20H             ;Page 1 char, set rentry bit (ORL)
00DF          = 251 ChOnP2  Equ     0DFH            ;Page 2 char, reset rentry bit (ANL)
0010          = 252 TstPrn  Equ     10H             ;Char print test
00EF          = 253 NrmPrn  Equ     0EFH            ;Normal char input
              = 254
0008          = 255 EOF     Equ     08H             ;set EOF Flag
00F7          = 256 ClrEOF  Equ     0F7H            ;clear EOF flag - Not EOF
0004          = 257 CRLF    Equ     04H             ;CR/LF
00FB          = 258 ClrCR   Equ     0FBH            ;Clear CR/LF
0002          = 259 CBFLn   Equ     02H             ;Full Line in Char Buffer
00FD          = 260 NCBFLn  Equ     0FDH            ;Not Full Line in Char Buffer
0001          = 261 IntCBR  Equ     01H             ;Init of CB registers done
00FE          = 262 ClICBR  Equ     0FEH            ;Init of CB registers not done
              = 263
              = 264
              = 265 ; PG
              = 266 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 267 ;               Equates  (cont)
              = 268 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 269
              = 270 ;       Misc
              = 271 ; ------------------------------------------------------------
0004          = 272 RLPShf  Equ     04H             ;R-to-L print lookup table addr shift
              = 273
0020          = 274 Ascii   Equ     20H             ;hex nmbr of first Ascii Char
007F          = 275 AscLst  Equ     7FH             ;hex nmbr of last Ascii Char
              = 276
00F3          = 277 CRCpl   Equ     0F3H            ;ASCII control code 2's complement
00F6          = 278 LFCpl   Equ     0F6H            ; "
00F4          = 279 FFCpl   Equ     0F4H            ; "
00E5          = 280 EscCpl  Equ     0E5H            ; "
00E0          = 281 AscCpl  Equ     0E0H            ; "
00C8          = 282 FTCpl   Equ     0C8H            ; "
000D          = 283 CR      Equ     0DH             ;Ascii code (hex)
0020          = 284 Space   Equ     20H             ;Ascii code (hex)
              = 285
00B1          = 286 LAsEnd  Equ     81H             ;Ascii End 2's cpl - test line start
0082          = 287 PAsEnd  Equ     82H             ;Ascii End 2's cpl - within line print
007F          = 288 AscStp  Equ     7FH             ;Ascii mask, strip off MSB
0042          = 289 PgLnCt  Equ     66              ;Page Line Count: Default = 66
00C4          = 290 PgLCpl  Equ     0C4H            ;Printed lines per page test
001B          = 291 EOFCpl  Equ     1BH             ;EOF ascii code cpl
              = 292
              = 293 ;       Loop count values
              = 294 ; ------------------------------------------------------------
```

```
0006          = 295 NDtCCt  Equ     06H             ;Normal Dot Column Count
000A          = 296 EDtCCt  Equ     0AH             ;Expanded Dot Column Count
0004          = 297 PHCnt1  EQU     04H             ;NUMBER OF SM PHASES ON INIT
              = 298
0004          = 299 ILFCnt  Equ     04              ;Init LF step/phz count
0024          = 300 LPI6p6  Equ     36              ;Lines Per Inch 6.6
001B          = 301 LPI8p8  Equ     27              ;Lines Per Inch 8.8
0018          = 302 LPI10   Equ     24              ;Lines Per Inch 10
              = 303
0001          = 304 LineCt  Equ     01              ;linefeed count
0042          = 305 FmFdCt  Equ     66              ;lines per formfeed count
0003          = 306 Status  EQU     03H             ;SEE BELOW FOR STATUS BYTE DEF.
              = 307                                 ;   TEST:  SET FOR CR STPR MTR CONTROL
              = 308
              = 309
              = 310 ; PG

              = 311 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 312 ;       TIMER VALUES - UPI Timer/Counter is UP Counter
              = 313 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 314 ;       12 MHz Clk timings
              = 315 ; -------------------------------------------------------------
0080          = 316 DLYCL   EQU     80H             ;DELAY COUNT Long
0030          = 317 DLYCS   EQU     30H             ;DELAY COUNT Short
00CC          = 318 DlyTim  EQU     256-52          ;TIME DELAY constant ~2.0mS
0000          = 319 FailTm  EQU     256-256         ;FailSafe TIME = ~17.0mS
00CC          = 320 CrTmr1  EQU     256-52          ;CR Stpr Mtr Phase TIME = ~2.08mS
00BA          = 321 CrTmr2  EQU     256-70          ;CR Stpr Mtr Phase TIME = ~2.40mS
0092          = 322 CrTmr3  EQU     256-110         ;CR Stpr Mtr Phase TIME = ~4.16mS
00C0          = 323 IntTm2  EQU     256-64          ;Init Stpr Mtr Phase TIME = ~2.40mS
0098          = 324 LFTMR1  EQU     256-104         ;LF Stpr Mtr Phase TIME = ~4.16mS
              = 325
              = 326 ;       I/O port bit masks
00DF          = 327 NotBsy  Equ     0DFH            ;Not Busy
0020          = 328 Busy    Equ     20H             ;Busy
00EF          = 329 Ack     Equ     0EFH            ;Ack
0010          = 330 ReSAck  Equ     10H             ;ReSet Ack
              = 331
              = 332 ;       Misc bit Masks
000C          = 333 StrpLF  Equ     0CH             ;Strip off all bits but LF Stpr Mtr
0003          = 334 StrpCR  Equ     03H             ;Strip off all bits but CR Stpr Mtr
              = 335
              = 336 ;       Print Head fires on low going edge of Trigger.
              = 337 ;         bit #9 in dot column is masked off, always:  P2, bit 6
0040          = 338 PTRGLO  EQU     40H             ;PH TRIGGER BIT - LOW
00C0          = 339 PTRGHI  EQU     0C0H            ;PH TRIGGER BIT - HIGH
              = 340
              = 341 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 342 ;       Stepper Motor Phase State Equates
              = 343 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 344
              = 345 ;       Stepper Motor Phase Shift Index Offset Offset
0000          = 346 FStCRP  EQU     00H             ;F CR stpr mtr phase data start addr
0003          = 347 RStCRP  EQU     03H             ;R CR stpr mtr phase data start addr
0008          = 348 STLFF   Equ     08H             ;Paper feed stpr mtr phase data start addr
              = 349
              = 350 ;       CARRIAGE STEPPER MOTOR PHASE EQUATES
              = 351 ;       Forward (1 thru 4)  &  Reverse (4 thru 1) :
0001          = 352 CRMFP1  EQU     01B             ;CR STPR MTR PHASE 1
0003          = 353 CRMFP2  EQU     11B             ;CR STPR MTR PHASE 2
0002          = 354 CRMFP3  EQU     10B             ;CR STPR MTR PHASE 3
0000          = 355 CRMFP4  EQU     00B             ;CR STPR MTR PHASE 4
              = 356
              = 357         ;LINE FEED STEPPER MOTOR PHASE EQUATES
              = 358 ;       Forward:
0004          = 359 LFMFP1  EQU     0100B           ;LF STPR MTR PHASE 1
000C          = 360 LFMFP2  EQU     1100B           ;LF STPR MTR PHASE 2
0008          = 361 LFMFP3  EQU     1000B           ;LF STPR MTR PHASE 3
0000          = 362 LFMFP4  EQU     0000B           ;LF STPR MTR PHASE 4
              = 363
              = 364 ; PG
```

```
                = 365 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 366 ;       STEPPER MOTOR SELECT & CONTROL [CURRENT LIMITING]
                = 367 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                = 368
                = 369 ;       PORT BIT ASSIGNMENT:
                = 370
                = 371 ;                \ \ \
                = 372 ;                S S S -
                = 373 ;                L C C
                = 374 ;                F R R
                = 375 ;                  B 1
                = 376 ;                  0 3
                = 377 ;                    2
                = 378 ;       ------------------
                = 379 ;                5 5 5 5
                = 380 ;                3 2 1 0
                = 381 ;       ------------------
                = 382 ; CODING:
                = 383 ;       SLF      0 1 1 0          06H
                = 384 ;       SCR80    1 0 0 0          0AH
                = 385 ;       SCR132   1 1 0 0          0CH
                = 386 ;       SMOFF    1 1 1 0          0EH
                = 387 ;                W/SCR80 & SCR132 '0' [BOTH SELECTED]
                = 388 ;                  DEFAULT IS TO 80 COL.
                = 389 ;                  [DO NOT KNOW WHETHER SCR80='0' WILL
                = 390 ;                  SELECT 80 COL ONLY] - REQUIRES TEST.
                = 391
0008            = 392 SCR80   EQU      08H              ;SELECT CR STPR MTR - 80 COL
                = 393                                   ;  w/LF STPR MTR OFF
000C            = 394 SCR132  EQU      0CH              ;SELECT CR STPR MTR - 132 COL
                = 395                                   ;  w/LF STPR MTR OFF
0006            = 396 SLF     EQU      06H              ;SELECT LF STPR MTR ON
                = 397                                   ;  w/CR STPR MTR OFF
000E            = 398 SMOFF   EQU      0EH              ;SELECT CR & LF STPR MTR OFF
                  399
                  400
                  401 ; PG
                  402 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                  403 ;                 MAIN PROGRAM BODY
                  404 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                  405
                  406 ;         Power On / Reset Program Entry
                  407
                  408 ;         PROGRAM START
                  409
0000              410           Org      00H
                  411
0000 0408         412 START:    JMP      RESET
                  413
                  414 ;         INPUT BUFFER FULL INTERRUPT CALL ENTRY AND VECTOR
0003              415           ORG      03H
0003 1425         416 IBFIV:    Call     IBFIS
0005 93           417           RETR
                  418 ;         TIMER OVERFLOW INTERRUPT CALL ENTRY AND VECTOR
0007              419           ORG      07H
0007 1429         420 TMRIV:    Call     TMRIS
0009 C5           421           SEL      RB0
000A 83           422           Ret
                  423
                  424 ;                 INITIALIZATION
                  425
000B 15           426 ReseT:    Dis      I
000C 35           427           Dis      TCntI
000D B40F         428           Call     InitAl           ;set all critical outputs inactive
000F B42F         429           Call     ClrDM            ;clear all data memory - 93H to 7FH
                  430                                    ;  do not clear RB0, RB1 or Stack
0011 B44B         431           Call     InitCR           ;CALL CR SM POWER ON INIT
0013 9400         432           Call     InitLF           ;CALL LF SM POWER ON INIT
                  433
                  434 ;                 MAIN PROGRAM LOOP
                  435 ;                 All program segments are called from here
                  436
0015 B422         437 Home:     Call     CRHome           ;Call Home CR routine -
                  438                                    ;  fixes logical and physical CR Home
0017 B400         439           Call     Defalt           ;set default register values
0019 142C         440 CBInpt:   Call     ESCBfF           ;Stat Switch / CB Input Service Test
                  441                                    ;  test for:  CB full/fill, LF, FF,
                  442                                    ;               Char Prnt Test
```

```
001B 3400          443 Repeat: Call    SMDriv          ;Call Forward Stpr Mtr Drive
001D 940D          444         Call    LFDriv          ;Call Linefeed Stpr Mtr Drive
001F D5            445         SEL     RB1
0020 FA            446         Mov     A,ChStR1        ;get the Char Status Register RB1
0021 7215          447         JB3     Home            ;jump to CR SM Home if EOF bit set
0023 0419          448         Jmp     CBInpt          ;loop to Char Buffer Input test
                   449
                   450 ; PG
                   451 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   452 ;                   Interrupt Service Routine
                   453 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   454
                   455 ; ---------------------------------------------------------------------------
                   456 ;                   Input Buffer Full Interrupt Service Routine
                   457 ; ---------------------------------------------------------------------------
                   458
                   459 IBFIS:
                   460 ; ---------------------------------------------------------------------------
                   461 ;       Acknowledge Char input and set Hold/Busy Active
0025 8A20          462         ORL     P2,#Busy        ;get & set DBB ACK/Busy Bits
0027 15            463         Dis     I               ;disable IBF interrupts
0028 93            464         Ret
                   465
                   466 ; ---------------------------------------------------------------------------
                   467 ;                   Timer / Counter Interrupt Service Routine
                   468 ; ---------------------------------------------------------------------------
                   469 ;       ITF interrupt service routine disables all intr during
                   470 ;         stpr mtr phase shifting
0029 15            471 TMRIS:  Dis     I               ;disable IBF interrupts
002A 35            472         Dis     TCntI           ;dicable ITF interrupts
002B 83            473         Ret
                   474
                   475 ; PG
                   476 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   477 ;       External Status Switch Check/Char. Buffer Fill
                   478 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   479 ESCBfF:                         ;Prep for normal character handling/input
002C D5            480         SEL     RB1
002D FA            481         Mov     A,ChStR1        ;get the character stat reg byte
002E 53EF          482         ANL     A,#NrmPrn       ;set normal character input
0030 AA            483         Mov     ChStR1,A        ;store the stat byte
0031 C5            484         SEL     RB0
                   485
                   486 ;       Test External Status Port
0032 0F            487         MovD    A,P7            ;get the stat switch port bits
0033 123D          488         JB0     FormFd          ;  service Formfeed
0035 3245          489         JB1     LinFd           ;  service Linefeed
0037 5249          490         JB2     ChrTst          ;  service Character TEST
0039 725E          491         JB3     OnLine          ;  service Char Buffer Check/Fill
003B 042C          492         Jmp     ESCBfF          ;Loop
                   493 ; ---------------------------------------------------------------------------
003D FA            494 FormFd: Mov     A,GStR20        ;get the status byte
003E 4304          495         ORL     A,#FrmFd        ;set the formfeed stat flag
0040 AA            496         Mov     GStR20,A        ;store trhe status byte
0041 940D          497         Call    LfDriv          ;do a formfeed
0043 042C          498         Jmp     ESCBfF
                   499 ; ---------------------------------------------------------------------------
0045 940D          500 LinFd:  Call    LfDriv           ;do a line drive
0047 042C          501         Jmp     ESCBfF
                   502 ; ---------------------------------------------------------------------------
0049 D5            503 ChrTst: SEL     RB1
004A FA            504         Mov     A,ChStR1        ;get the character stat reg byte
004B 4310          505         ORL     A,#TstPrn       ;set character test flag
004D AA            506         Mov     ChStR1,A        ;store the stat byte
004E B823          507         Mov     TmpR10,#PTAscS  ;load the psuedo Ascii code tmp reg addr
0050 F0            508         Mov     A,@TmpR10       ;get the inc'd ascii code
0051 03B1          509         ADD     A,#LAsEnd       ;test for code end
0053 9657          510         JNZ     AscCLd          ;if not code end jmp to load
                   511                                 ;if end restart ascii at begining
0055 B020          512         Mov     @TmpR10,#Ascii  ;store the ascii code start
0057 F0            513 AscCLd: Mov     A,@TmpR10       ;get the ascii code again
0058 AF            514         Mov     OpnR71,A        ;place in the empty register
0059 10            515         Inc     @TmpR10         ;Inc start ASCII char in data memory
005A B439          516         Call    PrnTst          ;call the DM load procedure
005C C5            517         SEL     RB0             ;reselect reg bank 0
005D 83            518         Ret
                   519 ; ---------------------------------------------------------------------------
```

```
005E D5          520 OnLine:  SEL     RB1              ;select char buffer registers
005F 05          521          EN      I                ;enable interrupts
0060 FA          522 CBfCk1:  Mov     A,ChStR1         ;get the Char Stat Byte
0061 3267        523          JB1     CBCkEx           ;if Chr Buf has full line exit
0063 146D        524 IBfCk:   Call    CBFill           ;read a char into Char Buffer
0065 0460        525          Jmp     CBFCk1           ;loop to Char Buf Ful test
0067 C5          526 CBCkEx:  SEL     RB0
0068 83          527          Ret
                 528
                 529 ; PG
                 530 ; ---------------------------------------------------------------------------
                 531 ;       Character Input
                 532 ; ---------------------------------------------------------------------------
                 533 ;        Input Buffer Full service routine:  test for Char buffer full-exit
                 534 ;          else load char into char buffer
0069 D5          535 IBFSrv:  SEL     RB1
006A FA          536          Mov     A,ChStR1         ;get the RB0 stat byte
006B 32EC        537          JB1     CBFull           ;if Do Not Print Bit Set - EXIT
006D 527C        538 CBFill:  JB2     CBPad            ;test for CB padding flag
                 539                                   ;  if not pad enable char input
                 540                                   ;  tell the host to send char's
006F 05          541          EN      I
0070 D6EC        542          JNIBF   CBF1Ex
                 543 ; ---------------------------------------------------------------------------
                 544 ;        Acknowledge Char input and set Hold/Busy Active
0072 FA          545          Mov     A,ChStR1         ;get the RB1 Char Stat Byte
0073 127C        546          JB0     SkpInt           ;test for CB has been Initialized
                 547 ; ---------------------------------------------------------------------------
                 548 ;        Init of all Char handling registers
0075 4301        549          ORL     A,#IntCBR        ;set CB Reg skip Initialization stat bit
0077 AA          550          Mov     ChStR1,A         ;save the altered stat byte
0078 B97F        551          Mov     CAdrR1,#FCBfSt   ;load char reg w/char bufr strt
007A BD50        552          Mov     CCntR1,#ChBfSz   ;load char cnt reg w/char bufr size
                 553 CBPad:
007C ED86        554 SkpInt:  DJNZ    CCntR1,LdChar    ;DECREMENT BUFFER SIZE
007E FA          555          Mov     A,ChStR1         ;get the status byte
007F 4302        556          ORL     A,#CBFLn         ;set Char Buffer Full Line stat bit
0081 53FB        557          ANL     A,#ClrCr         ;clear the CR/(LF) stat bit
0083 53FE        558          ANL     A,#C1ICBR        ;reset CB Init bit: init CB reg on entry
0085 AA          559          Mov     ChStR1,A         ;store the status byte
0086 FA          560 LdChar:  Mov     A,ChStR1         ;get the status byte
0087 52E1        561          JB2     CBPad1           ;CB not full but CR/LF previously
                 562                                   ;  received so pad CB
0089 9AEF        563          ANL     P2,#Ack          ;output DBB Ack low
008B 22          564          In      A,DBB            ;read the Char
008C 537F        565          ANL     A,#AscStp        ;strip off MSB
008E A8          566          Mov     TmpR10,A         ;temp save char
008F 8A10        567          ORL     P2,#ReSAck       ;output DBB ACK High
                 568 ; ---------------------------------------------------------------------------
                 569 ;        test for ASCII printable character
0091 03E0        570          ADD     A,#ASCCp1        ;test for Carriage Return
0093 F697        571          JC      AsciiC           ;jmp to service
0095 049C        572          Jmp     ChrChk
0097 97          573 AsciiC:  Clr     C                ;clear carry flag
0098 F8          574          Mov     A,TmpR10         ;get the char back
0099 A1          575          Mov     @CAdrR1,A        ;load data memory w/Char
009A 04E3        576          Jmp     IBFSrE
                 577 ; ---------------------------------------------------------------------------
                 578 ;        test for CR/LF:  if CR/LF Strip off LF and exit setting
                 579 ;          Char Buffer Init Stat bit
009C F8          580 ChrChk:  Mov     A,TmpR10         ;get the char back
009D 03F3        581          ADD     A,#CRCp1         ;test for Carriage Return
009F C6C3        582          JZ      CRChr            ;  if CR go service it
00A1 F8          583          Mov     A,TmpR10         ;get the char back
00A2 031B        584          ADD     A,#EOFCp1        ;test for End Of File
00A4 96AA        585          JNZ     ChrCk1           ;if not EOF jmp to CB Pad
00A6 F8          586          Mov     A,TmpR10         ;if EOF, place it in CB
00A7 A1          587          Mov     @CAdrR1,A        ;load data memory w/CR Char
00A8 04B9        588          Jmp     ExtSet           ;Exit
00AA F8          589 ChrCk1:  Mov     A,TmpR10         ;get the status byte
00AB 03F4        590          ADD     A,#FFCp1         ;test for FormFeed
00AD 96E1        591          JNZ     CBPad1           ;if not FF Pad the CB
00AF C5          592          SEL     RB0
00B0 FA          593          Mov     A,GStR20         ;get the status byte
00B1 4304        594          ORL     A,#FrmFd         ;set the formfeed flag
00B3 AA          595          Mov     GStR20,A         ;store the status byte
00B4 D5          596          SEL     RB1
00B5 FA          597          Mov     A,ChStR1         ;get the status byte
00B6 4304        598          ORL     A,#CRLF          ;set CRLF stat bit:  pad balance of CB
                 599                                   ;  with Spaces until fill
00B8 AA          600          Mov     ChStR1,A         ;store the status byte
00B9 FA          601 ExtSet:  Mov     A,ChStR1         ;get the status byte
```

```
00BA 4302          602             ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
00BC 53FB          603             ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
00BE 53FE          604             ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
00C0 AA            605             Mov     ChStR1,A        ;store the status byte
00C1 04EC          606             Jmp     CBF1Ex          ;Exit
                   607 ; ---------------------------------------------------------------------
                   608 ;       Store CR char read in LF char (assume its always there) and ignor it
00C3 F8            609 CRChr:      Mov     A,TmpR1O        ;get the char back
00C4 A1            610             Mov     @CAdrR1,A       ;load data memory w/CR Char
00C5 C5            611             SEL     RBO
00C6 1E            612             INC     LnCtRO          ;inc the line count
00C7 FE            613             Mov     A,LnCtRO        ;get the line count
00C8 03C4          614             Add     A,#PgLCpl       ;test for page feed ln cnt
                   615                                     ;  if LnCt => PgLnCt set formfeed flag
00CA E6DO          616             JNC     NoFmFd          ;if not at end of page skip
00CC FA            617             Mov     A,GStR2O        ;get the status byte
00CD 4304          618             ORL     A,#FrmFd        ;set the form feed status flag
00CF AA            619             Mov     GStR2O,A        ;save the status byte
00DO D5            620 NoFmFd:     SEL     RB1
00D1 05            621             En      I               ;enable the IBF service
00D2 9ADF          622             ANL     P2,#NotBsy      ;output a not busy to Host
00D4 D6D4          623 LFTest:     JNIBF   LFTest          ;loop to next char
00D6 9AEF          624             ANL     P2,#Ack         ;output DBB Ack low
00D8 22            625             In      A,DBB           ;get next Char - assume it's a LF
                   626                                     ;  and ignor it (LF is forced upon
                   627                                     ;  detection of CR at print time)
00D9 FA            628 SetPad:     Mov     A,ChStR1        ;get the status byte
00DA 4304          629             ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                   630                                     ;  with Spaces until fill
00DC AA            631             Mov     ChStR1,A        ;store the status byte
00DD BA10          632             ORL     P2,#ReSAck      ;output DBB ACK High
00DF 04E3          633             Jmp     IBfSrE          ;jmp to addr step & exit
                   634 ; ---------------------------------------------------------------------
                   635 ;       fill Char Buffer with space
00E1 B120          636 CBPad1:     Mov     @CAdrR1,#Space  ;load data memory w/Char
                   637 ; ---------------------------------------------------------------------
                   638 ;       step the char address test for CB full &/or pad
00E3 C9            639 IBFSrE:     DEC     CAdrR1          ;Decrement dat memory location
00E4 FA            640             Mov     A,ChStR1        ;get the status byte
00E5 32EC          641             JB1     CBFull          ;test for CB Full
00E7 52EC          642             JB2     CBF1Ex          ;test for CB pad - exit w/Busy set
                   643 ; ---------------------------------------------------------------------
                   644 ;       Set Busy Line Low - Not Busy
00E9 05            645             EN      I
00EA 9ADF          646             ANL     P2,#NotBsy      ;output a not busy to Host
                   647 ; ---------------------------------------------------------------------
                   648 ;       exit w/ Busy Still set high
                   649 CBFull:
00EC 83            650 CBF1Ex:     Ret
                   651
                   652 ; PG
                   653 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   654 ;       L-to-R/R-to-L Carriage Stepper Motor Drive
                   655 ;          and Line Printing
                   656 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   657
0100               658             ORG     100H
                   659
0100 3622          660 SMDriv:     JTO     RAccel          ;if Print Head at left drive right
                   661                                     ;  else drive left
                   662 ; F=============================================================
                   663 FAccel:                     ;L-to-R Accelerate Stepper Motor
                   664 ;       Set the Forward acceleration/drive Entry status bits
0102 FA            665             Mov     A,GStR2O        ;get the status byte
0103 53BF          666             ANL     A,#ClrSnk       ;set not at speed flag = 0
0105 53DF          667             ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0107 4380          668             ORL     A,#LRPrnt       ;set L-to-R prnt stat bit = 1
0109 4301          669             ORL     A,#Ready        ;set stpr mtr ready - Drive On
010B 53EF          670             ANL     A,#ADIntN       ;set A/D Init Not Done
010D AA            671             Mov     GStR2O,A        ;store the status byte
010E D5            672 CBRDir:     SEL     RB1
010F FA            673             Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0110 4380          674             ORL     A,#LRPrnt       ;Set L-to-R print bit
0112 AA            675             Mov     ChStR1,A        ;save the Char Stat byte
0113 C5            676             SEL     RBO
                   677
                   678 ;       Restore the phase register index addresses
0114 B821          679             Mov     TmpROO,#CPSAdr  ;get Phz Storage Addr psuedo reg
0116 FO            680             Mov     A,@TmpROO       ;  get stored CR last phase index addr
0117 AB            681             Mov     PhzR3O,A        ;place last LF phase index addr in Phz Reg
                   682
```

```
                        ;       Set up for next phase bit output before entering timing loops
0118 1B                 INC     PhzR30                  ;STEP PHASE DB ADDRESS
0119 FB                 MOV     A,PhzR30                ;CHECK THE PHASE COUNT REG
011A 521E               JB2     IAFZrP                  ;CHK FOR COUNT BIT ROLLOVER
011C 2440               JMP     SMDflt                  ;skip adr index reset
011E BB00               IAFZrP: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
0120 2440               Jmp     SMDflt
                        ;R=============================================================
                        RAccel:         ;R-to-L Accelerate Stepper Motor
                        ;---------------------------------------------------------------
                        ;       Set the Reverse acceleration/drive Entry status bits
0122 FA                 Mov     A,GStR20        ;get the status byte
0123 53BF               ANL     A,#ClrSnk       ;clear Print Ready bit
0125 53DF               ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0127 537F               ANL     A,#RLPrnt       ;set R-to-L prnt status bit
0129 4301               ORL     A,#Ready        ;set stpr mtr ready - Drive On
012B 53EF               ANL     A,#ADIntN       ;set A/D Init Not Done
012D AA                 Mov     GStR20,A        ;store the status byte
012E D5                 RCBRDr: SEL     RB1
012F FA                 Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0130 537F               ANL     A,#RLPrnt       ;Set R-to-L print bit
0132 AA                 Mov     ChStR1,A        ;save the Char Stat byte
0133 C5                 SEL     RB0
                        ;---------------------------------------------------------------
                        ;       Restore the  phase register index address
0134 BB21               Mov     TmpR00,#CPSAdr  ;get Phz Storage Addr psuedo reg
0136 F0                 Mov     A,@TmpR00       ;  get stored CR last phase index addr
0137 AB                 Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
                        ;       Set up for next phase bit output before entering timing loops
0138 CB                 Dec     PhzR30          ;STEP PHASE DB ADDRESS
0139 FB                 MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
013A 523E               JB2     IARZrP          ;CHK FOR COUNT BIT ROLLOVER
013C 2440               JMP     SMDflt
013E BB03               IARZrP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER

                        SMDflt:
                        ;---------------------------------------------------------------
                        ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                        ;         store the unused stpr mtr current phase bits
0140 BB22               Mov     TmpR00,#LPSAdr  ;get the CR phz storeage addr
0142 F0                 Mov     A,@TmpR00       ;get the byute stored there
0143 E3                 MovP3   A,@A            ;get the phz data byte
0144 BB20               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0146 A0                 Mov     @TmpR00,A       ;store Last Phase bits - inderect

                        ;       SetUp Stpr Mtr Time Constant
0147 BDBA               MOV     TConR0,#CrTmr2  ;Load time constant Reg

                        Select:                 ;Select the Stpr Mtr
0149 2308               MOV     A,#SCRB0        ;GET CR SM SELECT BITS
014B 3D                 MOVD    P5,A            ;SELECT SM [SCRB0]
                        ;---------------------------------------------------------------
                        ;       SetUp Stpr Mtr Phase Shift index address register
                        ;       Output next phase and init timer to Std Time constant
014C FD                 STRTT:  MOV     A,TConR0        ;get time constant from reg
014D 62                 MOV     T,A             ;load the timer
014E FB                 MOV     A,PhzR30        ;get the phz reg indirect addr index
014F E3                 MovP3   A,@A            ;do indirect get of phz bits

                        ;---------------------------------------------------------------
                        ;       patch together the CR last and LF next phase bits
0150 BB20               Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0152 40                 ORL     A,@TmpR00       ;patch together CR existing & new LF
0153 3C                 MOVD    P4,A            ;OUTPUT BITS
0154 55                 STRT    T               ;START TIMER

                        ;       At start of timing loop do all Stpr Mtr Accel/Decel or
                        ;         Character SetUp overhead
0155 740C               Call    ADPTst          ;call Accel/Decel/Print Test

                        ;       Set up for next phase bit output before entering timing loops
                        PNRdy1: ;test for forward / reverse phase start indirect index to load
0157 FA                 Mov     A,GStR20        ;store stat byte
0158 F264               JB7     Ac1F2

                        ; reverse:
                        ;       Set up for next phase bit output before entering timing loops
015A CB                 Dec     PhzR30          ;STEP PHASE DB ADDRESS
015B FB                 MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
015C 5260               JB2     ARZroP          ;CHK FOR COUNT BIT ROLLOVER
015E 2462               JMP     ARNxtP
0160 BB03               ARZroP: MOV     PhzR30,#RStCRP  ;ZERO CR SM PHASE REGISTER
0162 246C               ARNxtP: Jmp     ANxtPh
```

```
                        767
                        768 ; forward:
                        769 ;       Set up for next phase bit output before entering timing loops
0164 1B                 770 AclF2:  INC     PhzR30          ;STEP PHASE DB ADDRESS
0165 FB                 771         MOV     A,PhzR30        ;CHECK THE PHASE COUNT REQ
0166 526A               772         JB2     AFZroP          ;CHK FOR COUNT BIT ROLLOVER
0168 246C               773         JMP     ANxtPh          ;skip adr index reset
016A BB00               774 AFZroP: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
                        775 ANxtPh:
                        776 ; ---------------------------------------------------------------
                        777 ;       stage one timer loop - T occurs before Std timeout
                        778 ;         wait for time out
016C 1682               779 TLOOP2: JTF     FAILSF          ;JMP ON TIME OUT-t DOES NOT OCCUR 1ST
016E 5672               780         JT1     tCHK1           ;IS T HIGH-JMP TO tCHK
0170 246C               781         JMP     TLOOP2          ;LOOP FOR JT1 OR JTF
0172 00                 782 tCHK1:  NOP                     ;delay, then double check T signal
0173 5677               783         JT1     tTruW1          ;JUMP T TEST TRUE-WAIT FOR JTF
0175 246C               784         JMP     TLOOP2
                        785 tTruW1:
                        786 ;       test for Print Ready bit - was Print Head Fire Setup Done?
                        787 ;       insert acceleration time/store time count done/notdone flag bit
0177 FA                 788         Mov     A,GStR20        ;get the status byte - prep for prnt
0178 D27C               789         JB6     RdyPr2          ;if Ready Print bit set call PHFire
017A 247E               790         Jmp     SkpPHF          ;  else skip Print Head Fire
017C 74CA               791 RdyPr2: Call    PHFire          ;print head solenoid fire routine
                        792 PNRdy2:
                        793 SkpPHF:
017E 1698               794 tTruW2: JTF     NXTPHZ          ;JUMP TO SM ERROR
0180 247E               795         JMP     tTruW2          ; LOOP TO TLOOP3
                        796 ; ---------------------------------------------------------------
                        797 ;       Step into failsafe/startup timer setup - T does not
                        798 ;         occurs before Std Time timeout, load failsafe SM protection
                        799 ;         time and wait for failsafe timeout or T.  If T occurs
                        800 ;         output phase immediately after T verify.
0182 2300               801 FAILSF: MOV     A,#FailTm       ;LOAD TIMER W/~15.0mS
0184 62                 802         MOV     T,A             ;  SM PROTECTION TIMEOUT
0185 55                 803         STRT    T               ;START TIMER
                        804 ; ---------------------------------------------------------------
                        805 ;       set the Status bit for Store time test
0186 FA                 806         Mov     A,GStR20        ;get the status byte
0187 4308               807         ORL     A,#FSCTm        ;set Failsafe/constant time flag
0189 AA                 808         Mov     GStR20,A        ;store the status byte
018A 5690               809 TLOOP3: JT1     tCHK2           ;IS T HIGH
018C 16AC               810         JTF     DSLECT          ;IF TIME OUT GO SM ERROR
018E 248A               811         JMP     TLOOP3          ;LOOP UNTIL T HIGH OR T-OUT
0190 00                 812 tCHK2:  NOP                     ;WAIT
0191 5695               813         JT1     StrTm1          ;jump out and store elapsed time
0193 248A               814         JMP     TLOOP3          ;  JMP TO FAILSF LOOP
0195 65                 815 StrTm1: Stop    TCnt            ;stop the failSafe Timer
0196 42                 816         Mov     A,T             ;read the timer
0197 A1                 817         MOV     @TStrR0,A       ;Store the time read in indexed addr
                        818                                 ; - next entry to A/D Memorize Time
                        819                                 ; routine will add time constant to it
                        820
                        821 ;    Test is CR Stpr Mtr Drive is finished prior to next phase output
                        822 ; ---------------------------------------------------------------
                        823 NXTPHZ:
                        824 ;       test for forward / reverse phase start indirect index to load
0198 FA                 825         Mov     A,GStR20        ;store stat byte
0199 F2A7               826         JB7     FDrive
                        827 ; Reverse  --  test for Reverse Stpr Mtr Drive procedure exit
                        828 ;       ALWAYS drive the CR to the left most HOME position
019B 26AC               829         JNTO    EOLn            ;test if home position jmp stop
019D FA                 830         Mov     A,GStR20        ;get the status byte
019E 124C               831         JB0     StrtT           ;test Ready stat bit:
                        832                                 ;  if bit 0 = 1 then Print More
01A0 4302               833         ORL     A,#DoNotP       ;set the do not print flag
01A2 53BF               834         ANL     A,#ClrSnk       ;clear Print Ready bit
01A4 AA                 835         Mov     GStR20,A        ;save the status byte
01A5 244C               836         Jmp     StrtT           ;continue CR SM drive
                        837                                 ;  - only exit is HR
                        838 ; Forward  --  test for Forward Stpr Mtr Drive procedure exit
                        839 FDrive:
01A7 FA                 840         Mov     A,GStR20        ;get the status byte
01A8 124C               841         JB0     StrtT           ;test Ready stat bit:
                        842                                 ;  if bit 0 = 1 then Print More
01AA 24AC               843         Jmp     EOLn            ;  else jmp to End Of Line exit
                        844                                 ;jump to start timer again
                        845 DSLECT:
01AC 5437               846 EOLn:   Call    DeclSM          ;call Sptr Mtr Deceleration
                        847 ; ---------------------------------------------------------------
                        848 ;       test for forward / reverse phase start indirect index to load
01AE FA                 849         Mov     A,GStR20        ;store stat byte
01AF F2B3               850         JB7     FDrvFS          ;jmp to f drive flag set
```

```
01B1 53FD          851          ANL     A,#OkPrnt       ;reset print flag - Ok Print
                   852                                  ; only if printing R-to-L
                   853
                   854 ;        update the status byte
01B3 53BF          855 FDrvFS:  ANL     A,#ClrSnk       ;clear Print Ready bit
                   856                                  ;set the Status bit for Store time test
01B5 53DF          857          ANL     A,#NAtSpd       ;Clear At Print Speed Bit
01B7 AA            858          Mov     @StR2O,A        ;save the status byte
01B8 83            859          RET
                   860
                   861 ; PG
                   862 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   863 ;        Stepper Motor Accel. Time Storeage
                   864 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   865
0200               866          ORG     200H
                   867                                  ;  Entry has Gen Stat Byte in A
0200 920C          868 ADMmTS:  JB4     DADInt          ;is A/D init done - then jmp
                   869
                   870 ;        1st Entry initializes the A/D Time store working registers
0202 B92F          871          Mov     TStrRO,#SMBfSt  ;Load the Stpr Mtr Buffer Start Addr
0204 BC0B          872          Mov     CntR4O,#ASBfSz  ;Load the Buffer Size
0206 FA            873          Mov     A,@StR2O        ;get the status byte
0207 4310          874          ORL     A,#ADIntD       ;set not 1st Accel Entry Flag
0209 AA            875          Mov     @StR2O,A        ;store the status byte
020A 4436          876          Jmp     ADExit          ;exit - 1st entry has not generated
                   877                                  ;  a closed time window
                   878
                   879 ;        Step the A/D Store count
020C EC26          880 DADInt:  DJNZ    CntR4O,StorCt   ;dec Times to store count
                   881                                  ;if not O store the count
                   882                                  ;else at end-set done flag
020E FA            883          Mov     A,@StR2O        ;get the status byte
020F 4320          884          ORL     A,#AtSpdF       ;set at speed/no more to store flag
0211 AA            885          Mov     @StR2O,A        ;store the status byte
                   886
                   887 ;        Initialize Char Print Registers:  if printing enabled
0212 3226          888          JB1     StorCt          ;if Do Not Print stat bit set
                   889                                  ;  Skip the Char register init
                   890
                   891 ;        Initialize all Char Reg's
                   892 ;        Test for L-to-R (forward) or R-to-L (reverse) printing
0214 D5            893          SEL     RB1
0215 FA            894          Mov     A,ChStR1        ;get the status byte
0216 4340          895          ORL     A,#CHIntD       ;set Char Init Done flag - bypass
0218 AA            896          Mov     ChStR1,A        ;save the status byte
0219 F21F          897          JB7     LdCBR1          ;test Chr Stat Byte Returned
                   898                                  ;  if bit 7 = 1 then Print L-to-R
021B B92F          899 LdCBR:   Mov     CAdrR1,#RCBfIS  ;load char reg w/char bufr strt R-to-L
021D 4421          900          Jmp     LdCBR2
                   901
021F B980          902 LdCBR1:  Mov     CAdrR1,#FCBfIS  ;load char reg w/char bufr strt L-to-R
                   903
0221 BD51          904 LdCBR2:  Mov     CCntR1,#ChBfIS  ;load char cnt reg w/char bufr size
0223 BB01          905          Mov     CDtCR1,#01        ;set the chr dot column cnt
0225 C5            906          SEL     RBO
                   907
                   908 ;        Test for t > Tc or t < Tc
0226 722C          909 StorCt:  JB3     FailST          ;test for failsafe time switch
                   910
                   911 ;        t < Tc = store Time Constant in use
0228 FD            912          Mov     A,TConRO        ;Get time constant currently in use
0229 A1            913          Mov     @TStrRO,A       ;Memorize/Store the time - indirect addr
022A 4435          914          Jmp     ADPRet
                   915
                   916 ;        t > Tc = store Time Constant + FailSafe Time Elapsed
                   917 ;        [see Accel/Cnst Speed/Decel WaveForm]
                   918 ;        equation is:     Trd - FailSafe Time = Tx
                   919 ;                     => Trd + Cpl(FailSafe Time) = Tx
                   920 ;                        Tx + Tcnst = T
                   921 ;                        Store/Memorize T
                   922 ;
022C F1            923 FailST:  Mov     A,@TStrRO       ;get the stored time
022D 03C8          924          Add     A,#FTCpl        ;2's cpl add
022F 6D            925          Add     A,TConRO        ;Add: Time stored + Time constant
                   926                                  ;  currently in use
0230 A1            927          Mov     @TStrRO,A       ;Memorize/Store the time
                   928 ;        Reset the Status bit for Store time test
                   929
0231 FA            930          Mov     A,@StR2O        ;get the status byte
0232 53F7          931          ANL     A,#ClrFSC       ;reset Failsafe/constant time flag
                   932                                  ;  assumes entry via constant time
```

```
0234 AA          933           Mov     GStR20,A          ;store the status byte
0235 C9          934 ADPRet:   Dec     TStrR0            ;step the A/D time data store addr
0236 83          935 ADExit:   Ret
                 936
                 937 ; PG
                 938 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 939 ;         Carriage Stepper Motor Deceleration
                 940 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 941
                 942 DeclSM:
                 943 ;         SetUp the Deceleration registers
0237 B925        944           Mov     TStrR0,#SMBEnd    ;Load the Stpr Mtr Buffer End Addr
0239 BC0A        945           Mov     CntR40,#DSBfSz    ;Load the Buffer Size
023B FB          946           MOV     A,PhzR30          ;get phase index address
023C E3          947           MovP3   A,@A              ;get phase from indexed address
                 948 ;         patch together the CR last and LF next phase bits
023D B820        949           Mov     TmpR00,#LastPh    ;load Last Phz psuedo reg to Temp Reg
023F 40          950           ORL     A,@TmpR00          ;patch together CR existing & new LF
0240 3C          951           MOVD    P4,A              ;OUTPUT BITS
0241 F1          952 StrtTD:   MOV     A,@TStrR0         ;get time from indexed data memory
0242 62          953           MOV     T,A               ;load timer
0243 55          954           STRT    T                 ;START TIMER
0244 19          955           Inc     TStrR0            ;step the Memorized time addr index reg
                 956 ;         test for forward / reverse phase start indirect index to load
0245 FA          957           Mov     A,GStR20          ;store stat byte
0246 F252        958           JB7     DclF2
                 959
                 960 ; reverse:
                 961 ;         Set up for next phase bit output before entering timing loops
0248 CB          962           Dec     PhzR30            ;decrement the phase addr
0249 FB          963           MOV     A,PhzR30          ;Get the phz data addr
024A 524E        964           JB2     DRZroP            ;CHK FOR COUNT BIT ROLLOVER
024C 445A        965           JMP     DNxtPh
024E BB03        966 DRZroP:   MOV     PhzR30,#RStCRP    ;ZERO CR SM PHASE REGISTER
0250 445A        967           Jmp     DclR2
                 968
                 969 ; forward:
                 970 ;         Set up for next phase bit output before entering timing loops
0252 1B          971 DclF2:    Inc     PhzR30            ;increment the phase addr
0253 FB          972           MOV     A,PhzR30          ;Get the phz data addr
0254 5258        973           JB2     DZroPh            ;CHK FOR COUNT BIT ROLLOVER
0256 445A        974           JMP     DNxtPh            ;skip adr index reset
0258 BB00        975 DZroPh:   MOV     PhzR30,#FStCRP    ;ZERO CR SM PHASE REGISTER
                 976 DNxtPh:                             ;set up for next phase shift
025A FB          977 DclR2:    MOV     A,PhzR30          ;get phase index address
025B E3          978           MovP3   A,@A              ;get phase from indexed address
                 979 ;         patch together the CR last and LF next phase bits
025C B820        980           Mov     TmpR00,#LastPh    ;load Last Phz psuedo reg to Temp Reg
025E 40          981           ORL     A,@TmpR00          ;patch together CR existing & new LF
025F 1663        982 TLoopD:   JTF     NxtPD2            ;JMP ON TIME OUT TO NEXT PH
0261 445F        983           JMP     TLoopD            ;LOOP UNTIL TIME OUT
0263 3C          984 NxtPD2:   MOVD    P4,A              ;OUTPUT BITS
0264 EC41        985           DJNZ    CntR40,StrtTD     ;Exit Test
                 986
                 987 ;         Set Storeage of next phase data in psuedo addr. This insures
                 988 ;         next phase is sequence correct for stpr mtr drive direction
0266 B821        989 SetRN:    Mov     TmpR00,#CPSAdr    ;get Phz Storeage Addr psuedo reg
0268 FB          990           MOV     A,PhzR30          ;get Phz data
0269 A0          991           Mov     @TmpR00,A         ;store CR Next phase index addr
026A B478        992 DMExit:   Call    DlyLng
026C B490        993           Call    DeSlSM
026E 83          994           RET
                 995
                 996 ; PG
                 997 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 998 ;         Stepper Motor Phase Shift Definitions
                 999 ;            All program procedures call this data.
                1000 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1001
0300            1002           ORG     300H
                1003
                1004 ;         DEFINE PHASE ADDRESSES:
                1005 ;         THE PHASE DATA IS ENCODED TO THE ADDRESS CALLED DURING THE
                1006 ;         STPR MTR ENERGIZE SEQUENCE CORRESPONDING TO THE NEXT PHASE
                1007 ;         OF THE SEQUENCE REQUIRED.
                1008
                1009 ;         CARRAGE MOTOR ENCODING: FORWARD  -  LEFT-to-RIGHT
                1010 ;                                 REVERSE  -  RIGHT-to-LEFT
                1011
```

```
                        ;       Reverse direction ENCODING is the same bytes accessed in
                        ;         reverse direction
                        ;
0300 01                 DB      CRMFP1
0301 03                 DB      CRMFP2
0302 02                 DB      CRMFP3
0303 00                 DB      CRMFP4
                        ;
                        ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        ;
                        ;       LF MOTOR PHASE ENCODE & DECODE: FORWARD (CLOCKWISE)
                        ;       Forward direction ENCODING:
                        ;       ---------------------------------------------
                        ;
0308                    ORG     308H
                        ;
0308 04                 DB      LFMFP1
0309 0C                 DB      LFMFP2
030A 08                 DB      LFMFP3
030B 00                 DB      LFMFP4
                        ;
                        ;
                        1034 ; PG
                        ;
                        ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1036 ;       Accel/Decel / Character Handling Test
                        ;
                        ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        ;       TEST  >  Is CR Stpr Mtr At Speed ??
                        ;         Yes - SetUp do Character Processing
                        ;         No  - Calculate / Store the Acceleration Phase Shift Time (11)
                        ; ---------------------------------------------------------------------------
                        ;
030C FA                 ADPTst: Mov     A,GStR2O        ;get the status byte
030D B211               JB5     PHFSet          ;test if Stpr Mtr At Speed
                                                ;  jmp to Prnt Head Fire Setup
030F 4400               Jmp     ADMmTS          ;else Call Accel/Decel Memory Time Store
                        ;
                        ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1049 ;       Process Characters for Printing
                        ;
                        ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        ;
                        ;       Character dot matrix - normal char
                        ;       d = Dot Column
                        ;       b = Blank Column
                        ;
                        ;       b d d d d d
                        ;       (Char Matrix)
                        ;       0 0 0 0  b
                        ;       0 0 0 1  d
                        ;       0 0 1 0  d
                        ;       0 0 1 1  d
                        ;       0 1 0 0  d
                        ;       0 1 0 1  d
                        ; ---------------------------------------------------------------------------
                        ;
0311 266B               PHFSet: JNTO    Retrn           ;if R=0 not ready to print-exit
0313 326A               JB1     NPRet           ;if Do Not Print stat bit set - EXIT
0315 D21B               JB6     SinkSt          ;if bit previously set-skip setting it
0317 FA                 Mov     A,GStR2O        ;get the status byte
0318 4340               ORL     A,#SnkSet       ;set Prnt Ready Sink bit
031A AA                 Mov     GStR2O,A        ;save the status byte
031B D5                 SinkSt: SEL     RB1
031C FA                 Mov     A,ChStR1        ;get char status register addr
031D D23A               JB6     PageCk          ;test Char Init Done, 1 = Print Dot
                                                ;  0 = Get Char
                        ;
                        1077 ; PG
                        ; ---------------------------------------------------------------------------
                        ;       Call for Individual character processing:  mid line test if CR/(LF)
                        ; ---------------------------------------------------------------------------
                        GetChr:
                        ;       test for CR/(LF) if it is the test position in the line
031F F1                 CRChCk: Mov     A,@CAdrR1       ;get character
0320 03F3               ADD     A,#CRCpl        ;test for Carriage Return
0322 C626               JZ      CrLnCk          ;  if CR go service it
0324 6437               Jmp     AsciC1          ;if not CR Insert Space Char
0326 FA                 CRLnCk: Mov     A,ChStR1        ;get char status register addr
0327 F22B               JB7     HlfLn           ;test Chr Stat Byte Returned
                                                ;  if bit 7 = 1 then Print L-to-R
0329 6432               Jmp     SpFill          ;if R-to-L print skip exit upon CR detect
                        ; ---------------------------------------------------------------------------
```

```
                1092 ;       if L-to-R printing exit the line if less than 1/2 line printed
032B FD         1093 HlfLn:  Mov     A,CCntR1        ;load char cnt reg w/char bufr size
032C 03D9       1094         ADD     A,#HlfCpl       ;add the 2's cpl of 1/2 chr buf size
032E F632       1095         JC      LnPad           ;if CB>1/2 full set CR/LF stat bit for pad
                1096                                 ;If CB<1/2 set buffer full stat bit
0330 648A       1097         Jmp     MdLnEx          ;mid-line exit
                1098 SpFill:
0332 97         1099 LnPad:  Clr     C               ;clear carry flag
0333 2320       1100         Mov     A,#Space        ;insert a space char
0335 6438       1101         Jmp     ChIsrt          ;char inserted jmp over get char
                1102 ; ---------------------------------------------------------------------
0337 F1         1103 AsciC1: Mov     A,@CAdrR1       ;get character
0338 7498       1104 ChIsrt: Call    GChar1          ;call the char lookup/trns table
                1105 ; ---------------------------------------------------------------------
                1106 ;       fetch the char dot column data
                1107 PageCk:                         ;page test for balance of char
033A FA         1108         Mov     A,ChStR1        ;get the status byte
033B B241       1109         JB5     FxJmp1          ;fix jmp over page boundries
033D F4EB       1110         Call    ChrPg2          ;Ascii char 50 - 7F Hex
033F 6443       1111         Jmp     MtxTst          ;jump to Matrix Test
0341 D4F0       1112 FxJmp1: Call    ChrPg1          ;Ascii char 20 - 4F Hex
                1113                                 ;  fall thru to print matrix
                1114                                 ;  and CB count tests
                1115
                1116 ; PG
                1117 ; ---------------------------------------------------------------------
                1118 ;       test the Char dot column print matrix count and Char buffer count
                1119 ; ---------------------------------------------------------------------
0343 EB61       1120 MtxTst: DJNZ    CDtCR1,PrntDt   ;test for dot col or blank
                1121                                 ;status byte in A upon entry here
0345 FA         1122         Mov     A,ChStR1        ;get the status byte
0346 53BF       1123         ANL     A,#CIntND       ;set Char Init NotDone stat Flag
0348 AA         1124         Mov     ChStR1,A        ;store the status byte
0349 ED58       1125         DJNZ    CCntR1,NotLCh   ;dec char cnt-jmp if Not Last Char
034B 53FD       1126         ANL     A,#NCBFln       ;if 0 reset stat bit Not CB Full Line
034D 53FE       1127         ANL     A,#ClICBR       ;reset CB Reg Init Flag - do Init
034F AA         1128         Mov     ChStR1,A        ;save the status byte
                1129
0350 C5         1130         SEL     RB0
0351 FA         1131         Mov     A,GStR20        ;get Gen Status register addr
0352 53FE       1132         ANL     A,#NotRdy       ;clear the ready bit
0354 AA         1133         Mov     GStR20,A        ;store the General Status Byte
0355 D5         1134         SEL     RB1
0356 6468       1135         Jmp     Retrn           ;EXit
                1136
                1137 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                1138 ;         (see GChar1 ASCII char code translation procedure)
                1139 ; ---------------------------------------------------------------------
                1140 NotLCh:                         ;A contains LR/RL bit properly set
0358 FA         1141         Mov     A,ChStR1        ;get char status register addr
0359 F25E       1142         JB7     StpCh2          ;test Chr Stat Byte Returned
                1143                                 ;  if bit 7 = 1 then Print L-to-R
035B 19         1144 StpCh1: Inc     CAdrR1          ;Increment char data memory addr.
035C 6468       1145         Jmp     Retrn
035E C9         1146 StpCh2: Dec     CAdrR1          ;Decrement char data memory addr.
035F 6468       1147         Jmp     Retrn           ;  fall thru to Get Char
                1148
                1149
                1150 ; ---------------------------------------------------------------------
                1151 ;       Re-Entry Exit point for same char:
                1152 ;         (before returning step the matrix)
                1153 ; ---------------------------------------------------------------------
                1154 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                1155 ;         (see GChar1 ASCII char code translation procedure)
                1156 ; ---------------------------------------------------------------------
                1157
                1158 PrntDt:
0361 FA         1159 PrnDir: Mov     A,ChStR1        ;get char status byte
0362 F267       1160         JB7     StpCD2          ;test Chr Stat Byte Returned
                1161                                 ;  if bit 7 = 1 then Print L-to-R
0364 CC         1162 StpCD1: Dec     CDotR1          ;reverse step char dot col index
                1163                                 ;  addr if R-to-L print
0365 6468       1164         Jmp     Retrn           ;skip over L-to-R print addr inc
0367 1C         1165 StpCD2: INC     CDotR1          ;forward step char dot col index
                1166                                 ;  addr if L-to-R print
                1167                                 ;EXIT
                1168
                1169 ; PG
```

```
                        1170 ; ---------------------------------------------------------------------
                        1171 ;       Character Print SetUp Exit Procedures
                        1172 ; ---------------------------------------------------------------------
                        1173 ;       Clean Standard Exit
                        1174 ; ---------------------------------------------------------------------
0368 C5                 1175 Retrn:  SEL     RB0
0369 83                 1176         Ret                     ;EXIT - return w/ Reg Bank 0 Reset
                        1177
                        1178 ;       Do Not Print exit:  set Stpr Mtr drive routine count loop
036A D5                 1179 NPRet:  SEL     RB1
036B FA                 1180         Mov     A,ChStR1        ;get the status byte
036C F27C               1181         JB7     SkpNPI          ;test print direction
                        1182 ; Reverse
036E C5                 1183         SEL     RB0
036F FA                 1184         Mov     A,GStR20        ;get the status byte
0370 53BF               1185         ANL     A,#ClrSnk       ;reset the print ready bit- skips PHFire call
0372 83                 1186         Ret
                        1187 ; Forward
0373 D27C               1188         JB6     SkpNPI          ;test for first PHFSet entry reg init
                        1189 ;       Initialize register variables upon first entry
                        1190 ;         end of count clears char to print bit in status byte
0375 4340               1191         ORL     A,#ChIntD       ;set Char Reg Init Done stat bit
0377 AA                 1192         Mov     ChStR1,A        ;save the status byte
0378 B807               1193         Mov     TmpR10,#07H     ;load CR stpr mtr count during NoPrnt
037A 6488               1194         Jmp     NPExit
037C E888               1195 SkpNPI: DJNZ    TmpR10,NPExit
037E FA                 1196         Mov     A,ChStR1        ;get the status byte
037F 53BF               1197         ANL     A,#CIntND       ;reset - char init not done
0381 AA                 1198         Mov     ChStR1,A        ;save the status byte
0382 C5                 1199         SEL     RB0
0383 FA                 1200         Mov     A,GStR20        ;get Gen Status register addr
0384 53FE               1201         ANL     A,#NotRdy       ;clear the ready bit
0386 AA                 1202         Mov     GStR20,A        ;store the General Status Byte
0387 83                 1203 NSetEx: Ret
0388 C5                 1204 NPExit: SEL     RB0
0389 83                 1205         Ret
                        1206
                        1207 ;       Mid-Line Exit
                        1208 ; ---------------------------------------------------------------------
                        1209 ;       EXIT - if CR and not > 1/2 line done during L-to-R print
038A FA                 1210 MdLnEx: Mov     A,ChStR1        ;get the status byte
038B 53FD               1211         ANL     A,#NCBFln       ;if 0 reset stat bit Not CB Full Line
038D 53FE               1212         ANL     A,#ClICBR       ;reset CB Reg Init Flag - do Init
038F AA                 1213         Mov     ChStR1,A        ;save the status byte
0390 C5                 1214         SEL     RB0
0391 FA                 1215         Mov     A,GStR20        ;get the RB0 status byte
0392 4302               1216         ORL     A,#DoNotP       ;set the Do Not Print Flag(for RAccel)
0394 53BF               1217         ANL     A,#ClrSnk       ;reset the print ready bit-exit FAccel
0396 AA                 1218         Mov     GStR20,A        ;save the status byte
0397 83                 1219         Ret
                        1220
                        1221 ; PG
                        1222 ; ---------------------------------------------------------------------
                        1223 ;       Character Dot Generator Math
                        1224 ;         Look-up Table Page Vectoring
                        1225 ;         Print Head Firing
                        1226 ; ---------------------------------------------------------------------
                        1227
0398 AE                 1228 GCHAR1: MOV     StrCR1,A                ;STORE THE CHAR
                        1229
                        1230 ;  screen for printable char [char +(cpl 20 Hex + 1 = E0 Hex)]
0399 03E0               1231         ADD     A,#0E0H
039B F69F               1232         JC      PrntCh
039D 64C9               1233         jmp     CntlCh          ;jmp to control char lookup table
039F 97                 1234 PrntCh: Clr     C               ;clear carry flag
03A0 FE                 1235         Mov     A,StrCR1        ;get the char again
                        1236
                        1237 ;  screen for char page [char +(cpl 50 Hex + 1 = B0 Hex)]
                        1238 ;    if carry char on page 2 else page 1
03A1 03B0               1239         ADD     A,#0B0H
03A3 F6AE               1240         JC      Page2
                        1241
                        1242 ;       Page 1 Character -- ASCII 20 Hex thru 4F Hex
                        1243 ;         Correct offset for lookup table page
                        1244 ;         ((char + E0 Hex)*5 = Page 1 index addr)
                        1245 ; ---------------------------------------------------------------------
03A5 FA                 1246 Page1:  Mov     A,ChStR1        ;get the status byte
03A6 4320               1247         OrL     A,#ChDnP1       ;set the page rentry flag bit
03A8 AA                 1248         Mov     ChStR1,A        ;store the status byte
03A9 FE                 1249         Mov     A,StrCR1        ;get the char agian
03AA 03E0               1250         ADD     A,#0E0H         ;set page 1 relative 00 offset
03AC 64B8               1251         Jmp     Multi5          ;jump to address math function
                        1252
```

```
                1253 ;       Page 2 Character -- ASCII 20 Hex thru 4F Hex
                1254 ;         Correct offset for lookup table page two's complement
                1255 ;         of ASCII chr code LookUp Table page base char of 50H plus
                1256 ;         char * 5 ((char + B0 Hex)*5 = Page 2 index addr)
                1257 ; ---------------------------------------------------------------
03AE 97         1258 Page2:  Clr     C                  ;clear carry flag
03AF FA         1259         Mov     A,ChStR1           ;get the status byte
03B0 53DF       1260         AnL     A,#ChOnP2          ;set the page rentry flag bit
03B2 AA         1261         Mov     ChStR1,A           ;store the status byte
03B3 FE         1262         Mov     A,StrCR1           ;get the char agian
03B4 03B0       1263         ADD     A,#0B0H            ;set page 2 relative 00 offset
03B6 64BB       1264         Jmp     Multi5             ;fall thru to address math function
                1265
                1266 ;       Compute character page offset dot pattern index address
03B8 AE         1267 MULTI5: Mov     StrCR1,A           ;store the zero offset char
03B9 E7         1268         RL      A                  ;MULTIPLY CHR BY 5 TO
03BA E7         1269         RL      A                  ;  FIND THE ADDRESS
03BB 6E         1270         ADD     A,StrCR1           ;ADD 1 TO COMPLETE 5X
03BC AC         1271         MOV     CDotR1,A           ;SAVE THE ADDRESS
                1272
                1273 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                1274 ;         (see GChar1 ASCII char code translation procedure)
                1275 ; ---------------------------------------------------------------
03BD FA         1276         Mov     A,ChStR1           ;get char status byte
03BE F2C4       1277         JB7     LRPrn              ;test Chr Stat Byte Returned
                1278                                    ;  if bit 7 = 1 then Print L-toR
03C0 FC         1279         MOV     A,CDotR1           ;get the char index addr
03C1 0304       1280         ADD     A,#RLPShf          ;add char offset - start at end
                1281                                    ;of char, print it R-to-L
03C3 AC         1282         MOV     CDotR1,A           ;SAVE THE ADDRESS
                1283
                1284 ;       Set the status byte for Character SetUp done
                1285 ; ---------------------------------------------------------------
03C4 FA         1286 LRPrn:  Mov     A,ChStR1           ;get the status byte
03C5 4340       1287         ORL     A,#ChIntD          ;set 1st char col test bit = 0
03C7 AA         1288         Mov     ChStR1,A           ;store the status byte
03C8 83         1289         Ret                        ;return w/status byte in A
                1290 ;       test for non printable characters goes here
03C9 83         1291 Cntlch: Ret
                1292
                1293 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1294 ;       Print Head Fire
                1295 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1296
                1297 ;       Entry point for print head solenoid firing
                1298 ;         - test for status byte for dot/blank column position
03CA D5         1299 PHFire: SEL     RB1
03CB FB         1300         Mov     A,CDtCR1           ;set the chr dot column cnt
03CC 96D2       1301         JNZ     Fire               ;if char cnt not 0 - Fire Head Sol.
                1302                                    ;if Chr Dot Cnt 0, reset the
03CE BB06       1303 SetCnt: Mov     CDtCR1,#NDtCCt     ;  char dot column count
03D0 64DB       1304         Jmp     Retrn1             ;skip PH Fire
03D2 2340       1305 Fire:   MOV     A,#PTrgLo          ;get the Prnt Head Trigger byte
03D4 3A         1306         OUTL    P2,A               ;FIRE PRINT HEAD
03D5 23C0       1307         MOV     A,#PTrgHi          ;get the Prnt Head Trigger byte
03D7 3A         1308         OUTL    P2,A               ;FIRE PRINT HEAD
03D8 C5         1309 Retrn1: SEL     RB0
03D9 83         1310         Ret                        ;EXIT - return w/ Reg Bank 0 Reset
                1311
                1312 ; PG
                1313 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1314 ;       PaperFeed Stpr Mtr Drive
                1315 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1316
0400            1317         ORG     400H
                1318
                1319 ;       Init psuedo register with LF inderect addr start - subsequent
                1320 ;         exchanges of the psuedo register will yield correct value
0400 BC04       1321 InitLF: MOV     CntR40,#ILFCNT     ;INIT PHASE COUNT REG
0402 BB22       1322         Mov     TmpR00,#LPSAdr     ;get Phz Inderect Addr psuedo reg
0404 2308       1323         MOV     A,#StLFF           ;get LF starting addr
0406 A0         1324         Mov     @TmpR00,A          ;store LF phase index addr start
                1325                                    ;  in psuedo register
0407 BE01       1326         Mov     LnCtR0,#LineCt     ;set line count reg for 1 ln
                1327                                    ; enables exit following LF SM init
0409 B41B       1328         Jmp     LfDrv1             ;jump over line/form feed amd variable
                1329                                    ;  line spacing tests & setups
                1330
                1331 ;       LineFeed / FormFeed Drive
```

```
                        1332 ; ---------------------------------------------------------------------
                        1333
                        1334 ;       load step count constant for standard line spacing
                        1335 ; ---------------------------------------------------------------------
                        1336 ;       test for various line/inch spacing would go here
                        1337 ;         (and removal of constant setup below)
040B BC1B               1338         MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
                        1339
                        1340 ;       LineFeed/FormFeed Test
040D FA                 1341 LfDriv: Mov     A,GStR20        ;get the status byte
040E 5214               1342         JB2     FmFd            ;if linefeed jmp to cnt load
0410 BE01               1343 LnCtLd: Mov     LnCtR0,#LineCt  ;set line count reg for 1 line
0412 841B               1344         Jmp     LfDrv1          ;jmp to Start of Drive
0414 FE                 1345 FmFd:   Mov     A,LnCtR0        ;get the line count
0415 37                 1346         Cpl     A               ;2's cpl Line Count
0416 0301               1347         Add     A,#01
0418 0342               1348         Add     A,#PgLnCt       ;Add 2's cpl for Paging
                        1349                                 ;  PgLnCt - LnCt = n Lines to move
                        1350                                 ;  PgLnCt+(cpl(LnCt) = n lines to move
041A AE                 1351         Mov     LnCtR0,A        ;set the line count for FF
                        1352
                        1353 ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                        1354 ;         store the unused stpr mtr current phase bits
041B B821               1355 LFDrv1: Mov     TmpR00,#CPSAdr  ;get the CR phz storeage addr
041D F0                 1356         Mov     A,@TmpR00       ;get the byute stored there
041E E3                 1357         MovP3   A,@A            ;get the phz data byte
041F B820               1358         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0421 A0                 1359         Mov     @TmpR00,A       ;store Last Phase bits - inderect
                        1360 ;       exchange/store the phase register index addresses
0422 B822               1361         Mov     TmpR00,#LPSAdr  ;get Phz Inderect Addr psuedo reg
0424 F0                 1362         Mov     A,@TmpR00       ;get LF last phase index addr
0425 AB                 1363         Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
0426 BD98               1364         MOV     TConR0,#LFTMR1  ;Load time constant Reg
                        1365
                        1366 ;       Select the Stpr Mtr
0428 2306               1367         MOV     A,#SLF          ;GET CR SM SELECT BITS
042A 3D                 1368         MOVD    P5,A            ;SELECT SM [SCR80]
                        1369
                        1370 ; ---------------------------------------------------------------------
                        1371 ;       LineFeed / FormFeed Drive Loop
                        1372 ; ---------------------------------------------------------------------
042B FB                 1373         MOV     A,PhzR30        ;get the phz reg indirect addr index
042C E3                 1374         MovP3   A,@A            ;do indirect get of phz bits
                        1375 ;       patch together the CR last and LF next phase bits
042D B820               1376         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
042F 40                 1377         ORL     A,@TmpR00       ;patch together CR existing & new LF
                        1378 ;       start timer and step motor
0430 3C                 1379         MOVD    P4,A            ;OUTPUT BITS
                        1380
                        1381 StrtLF:
0431 FD                 1382 STRLFT: MOV     A,TConR0        ;get time constant from reg
0432 62                 1383         MOV     T,A             ;load the timer
0433 55                 1384         STRT    T               ;START TIMER
                        1385 ;       setup the next phase to output
0434 1B                 1386         INC     PhzR30          ;STEP PHASE DB ADDRESS
0435 FB                 1387         MOV     A,PhzR30        ;get the phase index address
0436 523A               1388         JB2     ZROPHL          ;test phase
0438 843C               1389         JMP     NXTPHL
043A BB08               1390 ZROPHL: MOV     PhzR30,#STLFF   ;re-init phase register
                        1391
043C FB                 1392 NXTPHL: MOV     A,PhzR30        ;get the phz reg indirect addr index
043D E3                 1393         MovP3   A,@A            ;do indirect get of phz bits
                        1394 ;       patch together the CR last and LF next phase bits
043E B820               1395         Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0440 40                 1396         ORL     A,@TmpR00       ;patch together CR existing & new LF
                        1397
0441 1645               1398 TLoopL: JTF     NXPHLF          ;jmp on time out to output nxt phz
0443 8441               1399         JMP     TLOOPL          ;loop until timer times out
                        1400
0445 3C                 1401 NXPHLF: MOVD    P4,A            ;step motor - OUTPUT BITS
0446 EC31               1402         DJNZ    CntR40,StrLFT   ;test for end of phase count for line
                        1403                                 ;prep for next line
                        1404
                        1405 ;       test for various line/inch spacing would go here
0448 BC1B               1406         MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
044A EE31               1407         DJNZ    LnCtR0,StrtLF   ;test for end of line count
                        1408
044C FA                 1409         Mov     A,GStR20        ;Get the status byte
044D 53FB               1410         ANL     A,#LineFd       ;reset for line feed
044F AA                 1411         Mov     GStR20,A        ;save the status byte
                        1412
                        1413 ;       store the phase register index addresses
                        1414 ;       Set LineFeed Stpr Mtr Next Phase index address
0450 B822               1415 SetLRN: Mov     TmpR00,#LPSAdr  ;get Phz Storage Addr psuedo reg
```

```
0452 FB          1416          Mov     A,PhzR30        ;get the phase index address
0453 A0          1417          Mov     @TmpR00,A       ;store LF Next phase index addr
0454 B478        1418          Call    DlyLng
0456 B490        1419          Call    DeSlSM
                 1420
                 1421 ;        Check if Char Buffer contains full line (80 char or nChar & CR)
                 1422 ;          exit otherwise continue to read in characters
                 1423 ;         Mov     A,GStR20        ;get the stat byte
                 1424 ;         JB1     ByPas1          ;if Do Not Print Bit Set - EXIT
                 1425 ;         Call    CBFck
0458 83          1426 ByPas1:  Ret
                 1427
                 1428 ; PG
                 1429 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1430 ;        Minor Software Subroutines
                 1431 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1432
0500             1433          ORG     500H
                 1434
                 1435 ; -------------------------------------------------------------
                 1436 ;        System initialization subroutines
                 1437 ; -------------------------------------------------------------
                 1438 Defalt:
                 1439 ; -------------------------------------------------------------
                 1440 ;        reset/set EOF status flag bit = 0
0500 D5          1441          SEL     RB1
0501 FA          1442          Mov     A,ChStR1        ;get the char status byte
0502 53F7        1443          ANL     A,#ClrEOF       ;clear the EOF flag bit
0504 AA          1444          Mov     ChStR1,A        ;store the char status byte
0505 B823        1445          Mov     TmpR10,#PTAscS  ;get the Ascii code tmp store addr
0507 B020        1446          Mov     @TmpR10,#Ascii  ;load the tmp stor reg w/ascii start
0509 C5          1447          SEL     RB0
                 1448 ; -------------------------------------------------------------
                 1449 ;        reset/set Ok-to-Print status flag bit = 0
050A FA          1450          Mov     A,GStR20        ;get the status byte
050B 53FD        1451          ANL     A,#OkPrnt       ;reset print flag - Ok Print
050D AA          1452          Mov     GStR20,A        ;save the status byte
050E 83          1453          RET
                 1454 InitAl:
                 1455 AllOff:
                 1456 ; -------------------------------------------------------------
                 1457 ;        CLEAR all outputs
050F C5          1458          SEL     RB0
0510 230F        1459          MOV     A,#0FH          ;FORCE PORT HI - R/ OF 555
0512 3E          1460          MOVD    P6,A
0513 23FF        1461          MOV     A,#0FFH         ;TURN ALL PRNT SOL's OFF
0515 39          1462          OUTL    P1,A
0516 23C0        1463          MOV     A,#PTRGHI       ;print head fire tirgger inactive
0518 3A          1464          OUTL    P2,A
0519 BA03        1465          ORL     P2,#03          ;set comm hdsk to ACK hi/Busy hi
051B BA00        1466          Mov     GStR20,#00H     ;clear the status registers
051D D5          1467          SEL     RB1
051E BA00        1468          Mov     ChStR1,#00H
0520 C5          1469          SEL     RB0
0521 83          1470          RET                     ;RETURN TO INIT ROUTINE
                 1471
                 1472 ; PG
                 1473 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1474 ;        Home Carriage / Print Head Assembly
                 1475 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1476
0522 FA          1477 CRHome:  Mov     A,GStR20        ;get the status byte
0523 4302        1478          ORL     A,#DoNotP       ;set the do not print flag
0525 AA          1479          Mov     GStR20,A        ;save the status byte
0526 362A        1480          JTO     RtoL            ;test for position of PH assembly
                 1481                                  ;  drive accordingly
0528 3402        1482          Call    FAccel          ;drive CR Stpr Mtr
052A 3422        1483 RtoL:    Call    RAccel          ;find the logical left home CR position
                 1484                                  ;delay a long time before continuing
052C B474        1485          Call    DlyVLg
052E 83          1486          Ret
                 1487
                 1488 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1489 ;        Clear Data Memory
                 1490 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1491
                 1492 ;        At PowerUp or Reset, following CR & LF Stpr Mtr Init, this
                 1493 ;        procedure clears data memory above RB0, Stack and RB1.
052F B87F        1494 ClrDM:   MOV     R0,#DMTop       ;GET BUFFER START LOCATION [HEX]
0531 B95D        1495          MOV     R1,#DMSIZE
0533 B000        1496 ClrDM1:  MOV     @R0,#00H        ;ZERO MEMORY LOCATION
```

```
0535 C8          1497          DEC     R0
0536 E933        1498          DJNZ    R1,ClrDM1       ;dec buffer, loop if not zero[end]
0538 83          1499          RET                     ;RETURN TO INIT ROUTINE
                 1500
                 1501 ; PG
                 1502 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1503 ;         Character Print TEST
                 1504 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1505
                 1506 PrnTst:
                 1507 ;   TEST ---  load the char buffer with successive increments of
                 1508 ;             the ascii code start.  test for end of ascii
                 1509 ;             printable chars and reinit the char stream loaded.
                 1510
0539 B97F        1511 CTInt:   Mov      CAdrR1,#FCBfSt  ;load char reg w/char bufr strt
053B BD50        1512          Mov      CCntR1,#ChBfSz  ;load char cnt reg w/char bufr size
                 1513 ChTst:                           ;Test char buffer fill with ASCII Char Code
053D FF          1514          Mov      A,opnr71        ;get the ascii char
053E A1          1515          Mov      @CAdrR1,A       ;load data memory w/Char
053F C9          1516          DEC      CAdrR1          ;Decrement dat memory location
0540 1F          1517          INC      opnr71          ;Increment Ascii char number
0541 0382        1518          ADD      A,#PAsEnd       ;test for ascii code end
0543 9647        1519          JNZ      ChrTGo          ;if not end jmp over code restart
0545 BF20        1520          Mov      OpnR71,#Ascii
0547 ED3D        1521 ChrTGo:  DJNZ     CCntR1,ChTst    ;dec buffer, loop if not zero[end]
0549 C5          1522          SEL      RB0
054A 83          1523          RET                     ;ELSE RETURN TO INIT ROUTINE
                 1524
                 1525 ; PG
                 1526 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1527 ;         CR Stpr Mtr Power On Initialization and
                 1528 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1529 ;         This routine drives the CR or LF stpr mtr for four phase
                 1530 ;           shifts for initialization.
                 1531 INITCR:                            ;POWER ON INIT STPR MTR
054B BC04        1532          MOV      CntR40,#PhCnt1  ;load phase cnt reg for INIT
054D 2308        1533          MOV      A,#SCR80        ;GET CR SM SELECT BITS
054F 3D          1534          MOVD     P5,A            ;SELECT SM [SCR80]
0550 BDC0        1535          MOV      TConR0,#IntTm2  ;Load time constant Reg
0552 BB00        1536          MOV      PhzR30,#FStCRP  ;zero SM phase reg'- forward
0554 FB          1537          MOV      A,PhzR30        ;get phase index register byte
0555 E3          1538          MovP3    A,@A            ;load indexed phase shift byte
0556 3C          1539          MOVD     P4,A            ;OUTPUT BITS
0557 FD          1540 STRTTR:  MOV      A,TConR0        ;GET TIMER CONSTANT
0558 62          1541          MOV      T,A
0559 55          1542          STRT     T               ;START TIMER
055A 1B          1543          INC      PhzR30          ;step phase index register
055B FB          1544          MOV      A,PhzR30        ;CHECK THE PHASE COUNT REG
055C 5260        1545          JB2      ZroRg2
055E A462        1546          JMP      NxtPhR
0560 BB00        1547 ZroRg2:  MOV      PhzR30,#FStCRP  ;zero SM phase reg - forward
                 1548 NxtPhR:
0562 FB          1549          MOV      A,PhzR30        ;get phase index register byte
0563 E3          1550          MovP3    A,@A            ;load indexed phase shift byte
0564 1669        1551 TLoopR:  JTF      NXPHR1          ;JMP ON TIME OUT TO NEXT PH
0566 A464        1552          JMP      TLoopR          ;LOOP UNTIL TIME OUT
0568 3C          1553          MOVD     P4,A            ;OUTPUT BITS
0569 EC57        1554 NXPHR1:  DJNZ     CntR40,STRTTR
                 1555 ; ---------------------------------------------------------------------
                 1556 ;         store the last phase register index addresses
056B B821        1557          Mov      TmpR00,#CPSAdr  ;get Phz Storage Addr psuedo reg
056D FB          1558          Mov      A,PhzR30        ;place last CR phase index addr in Phz Reg
056E A0          1559          Mov      @TmpR00,A       ;  store CR last phase index addr
056F B478        1560          Call     DlyLng
0571 B490        1561          Call     DeSlSM
0573 83          1562          RET
                 1563
                 1564 ; PG
                 1565 ; ---------------------------------------------------------------------
                 1566 ;         Time Delay Subroutines
                 1567 ; ---------------------------------------------------------------------
                 1568
                 1569 ;         Very Long
0574 B87F        1570 DlyVLg:  MOV      TmpR00,#7FH     ;LOAD DELAY COUNT IN REG.
0576 A47E        1571          Jmp      DlyST
                 1572
                 1573 ;         Long
0578 B880        1574 DlyLng:  MOV      TmpR00,#DlyCL   ;LOAD DELAY COUNT IN REG.
057A A47E        1575          Jmp      DlyST
                 1576
```

```
                1577 ;          Not So Long - Short
057C B830       1578 DlySht:    MOV     TmpR00,#DlyCS   ;LOAD DELAY COUNT IN REG.
                1579
                1580 ;          Start Delay
057E 23CC       1581 DlyST:     MOV     A,#DlyTim       ;GET MAX TIMER DELAY
0580 62         1582 NxtTLd:    MOV     T,A             ;LOAD TIMER
0581 55         1583            STRT    T               ;START TIMER
                1584
0582 168D       1585 DlyLop:    JTF     DlyTO           ;LOOP
                1586
                1587 ;          Char buffer fill during time loop:
0584 D5         1588            SEL     RB1
0585 FA         1589            Mov     A,ChStR1        ;get the character stat reg byte
0586 928A       1590            JB4     SkpCI           ;  test for normal char input
                1591                                    ;  or skip if char prnt test
0588 1469       1592            Call    IBFSrv          ;service the char buffer fill
058A C5         1593 SkpCI:     SEL     RB0
058B A482       1594            JMP     DlyLOP                                          *
058D E880       1595 DlyTO:     DJNZ    TmpR00,NxtTLd   ;dec delay count & test for exit
058F 83         1596            RET
                1597 ; ---------------------------------------------------------------------
                1598 ;          Stpr Mtr Deselect
                1599 ; ---------------------------------------------------------------------
                1600 ;          Stepper Motor DeSelect Routine
                1601 DESLSM:                            ;DESELECT LF/CR SM
0590 230E       1602 SMEROR:    MOV     A,#SMOFF        ;GET LF/CR SM DE-SELECT BITS
0592 3D         1603            MOVD    P5,A            ;DE-SELECT CR SM
0593 83         1604            RET
                1605
                1606 $INCLUDE(:F1:CHRTBL.OV1)
               =1607
               =1608 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               =1609 ;          Character Dot Generator Look-up Table Page 1
               =1610 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               =1611
               =1612
               =1613 ;                  Character Table Page 1, contains
               =1614
               =1615 ;       20H ----------------------------------------> 4FH
               =1616
               =1617 ;     " (sp)!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM "
               =1618
               =1619 ; ---------------------------------------------------------------------
               =1620
0600           =1621            ORG     600H
               =1622
               =1623 ; ---------------------------------------------------------------------
               =1624 ;          Page 1  --  Character Dot Pattern Fetch
               =1625 ;          <<<  actual assembled character table code not listed  >>>
               =1626 ; ---------------------------------------------------------------------
               =1627 $NoList
               =1676 $List
               =1677 ;          Listing below is for reference only, actual code is not listed
               =1678 ;            at assembly time.
               =1679
               =1680 ; ---------------------------------------------------------------------
               =1681 ; asc20:          DB      7FH, 7FH, 7FH, 7FH, 7FH         ;SPACE
               =1682 ; asc21:          DB      7FH, 7FH, 20H, 7FH, 7FH         ;!
               =1683 ; asc22:          DB      7FH, 7FH, 78H, 7FH, 78H         ;"
               =1684 ; asc23:          DB      6BH, 00H, 6BH, 00H, 6BH         ;#
               =1685 ; asc24:          DB      5BH, 55H, 00H, 55H, 6DH         ;$
               =1686 ; asc25:          DB      5CH, 6CH, 77H, 1BH, 1DH         ;%
               =1687 ; asc26:          DB      19H, 26H, 26H, 59H, 2FH         ;&
               =1688 ; asc27:          DB      7FH, 7FH, 7CH, 7FH, 7FH         ;'
               =1689 ; asc28:          DB      63H, 5DH, 3EH, 7FH, 7FH         ;(
               =1690 ; asc29:          DB      7FH, 7FH, 3EH, 5DH, 63H         ;)
               =1691 ; asc2A:          DB      5DH, 6BH, 00H, 6BH, 5DH         ;*
               =1692 ; asc2B:          DB      77H, 77H, 41H, 77H, 77H         ;+
               =1693 ; asc2C:          DB      7FH, 3FH, 4FH, 7FH, 7FH         ;,
               =1694 ; asc2D:          DB      77H, 77H, 77H, 77H, 77H         ;-
               =1695 ; asc2E:          DB      7FH, 1FH, 1FH, 7FH, 7FH         ;.
               =1696 ; asc2F:          DB      5FH, 6FH, 77H, 7BH, 7DH         ;/
               =1697 ; asc30:          DB      41H, 2EH, 36H, 3AH, 41H         ;0
               =1698 ; asc31:          DB      7FH, 3DH, 00H, 3FH, 7FH         ;1
               =1699 ; asc32:          DB      3DH, 1EH, 2EH, 36H, 39H         ;2
               =1700 ; asc33:          DB      5DH, 3EH, 36H, 36H, 49H         ;3
               =1701 ; asc34:          DB      67H, 6BH, 6DH, 00H, 6FH         ;4
               =1702 ; asc35:          DB      58H, 3AH, 3AH, 3AH, 46H         ;5
               =1703 ; asc36:          DB      43H, 35H, 36H, 36H, 4EH         ;6
               =1704 ; asc37:          DB      7EH, 0EH, 76H, 7AH, 7CH         ;7
               =1705 ; asc38:          DB      49H, 36H, 36H, 36H, 49H         ;8
```

```
=1706 ; asc39:          DB      39H, 36H, 36H, 56H, 61H         ;9
=1707 ; asc3A:          DB      7FH, 7FH, 6BH, 7FH, 7FH         ;:
=1708 ; asc3B:          DB      7FH, 3FH, 4BH, 7FH, 7FH         ;;
=1709 ; asc3C:          DB      77H, 6BH, 5DH, 3EH, 7FH         ;<
=1710 ; asc3D:          DB      6BH, 6BH, 6BH, 6BH, 6BH         ;=
=1711 ; asc3E:          DB      7FH, 3EH, 5DH, 6BH, 77H         ;>
=1712 ; asc3F:          DB      79H, 7EH, 26H, 7AH, 7DH         ;?
=1713 ; asc40:          DB      41H, 3EH, 22H, 36H, 71H         ;@
=1714 ; asc41:          DB      03H, 6DH, 6EH, 6DH, 03H         ;A
=1715 ; asc42:          DB      00H, 36H, 36H, 36H, 49H         ;B
=1716 ; asc43:          DB      41H, 3EH, 3EH, 3EH, 5DH         ;C
=1717 ; asc44:          DB      00H, 3EH, 3EH, 5DH, 63H         ;D
=1718 ; asc45:          DB      00H, 36H, 36H, 36H, 36H         ;E
=1719 ; asc46:          DB      00H, 76H, 76H, 76H, 76H         ;F
=1720 ; asc47:          DB      41H, 3EH, 3EH, 2EH, 0DH         ;G
=1721 ; asc48:          DB      00H, 77H, 77H, 77H, 00H         ;H
=1722 ; asc49:          DB      7FH, 3EH, 00H, 3EH, 7FH         ;I
=1723 ; asc4A:          DB      5FH, 3FH, 3FH, 3FH, 40H         ;J
=1724 ; asc4B:          DB      00H, 77H, 6BH, 5DH, 3EH         ;K
=1725 ; asc4C:          DB      00H, 3FH, 3FH, 3FH, 3FH         ;L
=1726 ; asc4D:          DB      00H, 7DH, 73H, 7DH, 00H         ;M
=1727 ; asc4E:          DB      0aaH, 0dfH, 0efH, 0f7H, 0aaH    ;test
=1728 ; asc4F:          DB      55H, 0dfH, 0efH, 0f7H, 55H      ;test
=1729 ; asc4E:          DB      00H, 7BH, 77H, 6FH, 00H         ;N
=1730 ; asc4F:          DB      41H, 3EH, 3EH, 3EH, 41H         ;O
=1731 ; ---------------------------------------------------------------------------
=1732 ;         End Page 1  --  Character Dot Pattern Fetch
=1733
=1734 ; ---------------------------------------------------------------------------
=1735 ;         Character Dot Pattern Fetch
=1736 ; ---------------------------------------------------------------------------
=1737
06F0 FC     =1738 ChrPg1: MOV     A,CDotR1          ;get char index address offset
06F1 A3     =1739         MOVP    A,@A              ;get column dot patern byte
            =1740
            =1741 ;       this bit fix necessary to not underline each character
            =1742 ;         this saves fixing each bit in the look up table
            =1743
06F2 4380   =1744         ORL     A,#80H            ;char bit fix
06F4 39     =1745         OutL    P1,A              ;output the dot pattern
06F5 83     =1746         RET                       ;exit with byte in acc
            =1747
            =1748 ; ---------------------------------------------------------------------------
            =1749 ;       END Page 1  --  Character Dot Pattern Fetch
            =1750 ; ---------------------------------------------------------------------------
            =1751
            =1752
            =1753 ;       PAGE 2  -- Character Dot Generator Look-Up Table
            =1754 ; ---------------------------------------------------------------------------
            =1755
            =1756 ;               Character Table Page 2, contains
            =1757
            =1758 ;     50H ----------------------------------------------> 7EH
            =1759
            =1760 ;   " NOPQRSTUVWXYZ[\]^_(?)abcdefghijklmnopqrstuvwxyz{|}~ "
            =1761
            =1762 ; ---------------------------------------------------------------------------
            =1763
0700        =1764         ORG     700H
            =1765
            =1766 ; ---------------------------------------------------------------------------
            =1767 ;       Page 2  --  Character Dot Pattern Fetch
            =1768 ;       <<<  Actual assembled character table code not listed  >>>
            =1769 ; ---------------------------------------------------------------------------
            =1770 $NoLIST
            =1818 $List
            =1819 ;       Listing below is for reference only, actual code is not listed
            =1820 ;         at assembly time.
            =1821
            =1822 ; ---------------------------------------------------------------------------
            =1823 ; asc50:          DB      00H, 76H, 76H, 76H, 79H         ;P
            =1824 ; asc51:          DB      41H, 3EH, 2EH, 5EH, 21H         ;Q
            =1825 ; asc52:          DB      00H, 76H, 66H, 56H, 39H         ;R
            =1826 ; asc53:          DB      59H, 36H, 36H, 36H, 4DH         ;S
            =1827 ; asc54:          DB      7EH, 7EH, 00H, 7EH, 7EH         ;T
            =1828 ; asc55:          DB      40H, 3FH, 3FH, 3FH, 40H         ;U
            =1829 ; asc56:          DB      60H, 5FH, 3FH, 5FH, 60H         ;V
            =1830 ; asc57:          DB      00H, 5FH, 67H, 5FH, 00H         ;W
            =1831 ; asc58:          DB      1CH, 6BH, 77H, 6BH, 1CH         ;X
            =1832 ; asc59:          DB      7CH, 7BH, 07H, 7BH, 7CH         ;Y
```

```
                =1833 ; asc5A:          DB      1EH, 2EH, 36H, 3AH, 3CH         ; Z
                =1834 ; asc5B:          DB      00H, 3EH, 3EH, 3EH, 7FH         ; [
                =1835 ; asc5C:          DB      7DH, 7BH, 77H, 6FH, 5FH         ; \
                =1836 ; asc5D:          DB      7FH, 3EH, 3EH, 3EH, 00H         ; ]
                =1837 ; asc5E:          DB      6FH, 77H, 7BH, 77H, 6FH         ; ^
                =1838 ; asc5F:          DB      3FH, 3FH, 3FH, 3FH, 3FH         ; _
                =1839 ; asc60:          DB      7DH, 7BH, 77H, 0FFH, 0FFH       ; \
                =1840 ; asc61:          DB      0DFH, 0ABH, 0ABH, 0ABH, 087H    ; a
                =1841 ; asc62:          DB      080H, 0B7H, 0B7H, 0B7H, 0CFH    ; b
                =1842 ; asc63:          DB      0C7H, 0BBH, 0BBH, 0BBH, 0BBH    ; c
                =1843 ; asc64:          DB      0CFH, 0B7H, 0B7H, 0B7H, 080H    ; d
                =1844 ; asc65:          DB      0C7H, 0ABH, 0ABH, 0ABH, 0B7H    ; e
                =1845 ; asc66:          DB      0F7H, 0B1H, 0F6H, 0FEH, 0FDH    ; f
                =1846 ; asc67:          DB      0F7H, 0ABH, 0ABH, 0ABH, 0C3H    ; g
                =1847 ; asc68:          DB      080H, 0F7H, 0FBH, 0FBH, 087H    ; h
                =1848 ; asc69:          DB      0FFH, 0BFH, 0BBH, 0BFH, 0FFH    ; i
                =1849 ; asc6A:          DB      0DFH, 0BFH, 0BBH, 0C2H, 0FFH    ; j
                =1850 ; asc6B:          DB      0FFH, 080H, 0EFH, 0D7H, 0BBH    ; k
                =1851 ; asc6C:          DB      0FFH, 0BEH, 080H, 0BFH, 0FFH    ; l
                =1852 ; asc6D:          DB      087H, 0FBH, 0E7H, 0FBH, 087H    ; m
                =1853 ; asc6E:          DB      083H, 0F7H, 0FBH, 0FBH, 087H    ; n
                =1854 ; asc6F:          DB      0C7H, 0BBH, 0BBH, 0BBH, 0C7H    ; o
                =1855 ; asc70:          DB      084H, 0EBH, 0EBH, 0EBH, 0F7H    ; p
                =1856 ; asc71:          DB      0F7H, 0EBH, 0EBH, 0EBH, 084H    ; q
                =1857 ; asc72:          DB      0FFH, 083H, 0F7H, 0FBH, 0FBH    ; r
                =1858 ; asc73:          DB      087H, 0ABH, 0ABH, 0ABH, 0DBH    ; s
                =1859 ; asc74:          DB      0FBH, 0C1H, 0BBH, 0DFH, 0FFH    ; t
                =1860 ; asc75:          DB      0C3H, 0BFH, 0BFH, 0BFH, 0C3H    ; u
                =1861 ; asc76:          DB      0E3H, 0DFH, 0BFH, 0DFH, 0E3H    ; v
                =1862 ; asc77:          DB      0C3H, 0BFH, 0CFH, 0BFH, 0C3H    ; w
                =1863 ; asc78:          DB      0BBH, 0C7H, 0EFH, 0C7H, 0BBH    ; x
                =1864 ; asc79:          DB      0FFH, 0B3H, 0AFH, 0AFH, 0C3H    ; y
                =1865 ; asc7A:          DB      0BBH, 09BH, 0ABH, 0B3H, 0BBH    ; z
                =1866 ; ASC7B:          DB      07FH, 077H, 049H, 03EH, 03EH    ; {
                =1867 ; ASC7C:          DB      0FFH, 0FFH, 088H, 0FFH, 0FFH    ; |
                =1868 ; ASC7D:          DB      03EH, 03EH, 009H, 077H, 07FH    ; }
                =1869 ; ASC7E:          DB      067H, 07BH, 067H, 05FH, 067H    ; ~
                =1870
                =1871 ; ---------------------------------------------------------------
                =1872 ;         Character Dot Pattern Fetch
                =1873 ; ---------------------------------------------------------------
                =1874
07EB FC         =1875 ChrPg2:   MOV     A,CDotR1        ;get char index address offset
07EC A3         =1876           MOVP    A,@A            ;get column dot patern byte
                =1877
                =1878 ;         this bit fix necessary to not underline each character
                =1879 ;           this saves fixing each bit in the look up table
                =1880
07ED 4380       =1881           ORL     A,#80H          ;char bit fix
07EF 39         =1882           OutL    P1,A            ;output the dot pattern
07F0 83         =1883           RET                     ;exit with byte in acc
                 1884
                 1885
                 1886
                 1887 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1888 ;                 Program End
                 1889 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1890
                 1891            END

ASSEMBLY COMPLETE,   NO ERRORS
```

APPENDIX B. SOFTWARE PRINTER ENHANCEMENTS

This section describes several software enhancements which could be implemented as additions to the software developed for this Application Note. Space is available for most of the items described. Approximately 5 bytes of Data Memory would be required to implement most of the features. Two bytes would be used for status flags, and two bytes for temporary data or count storage. It is possible to use less than five bytes, but this would require the duplicate use of some flags, or other Data Memory storage, which will significantly complicate the software coding and debug tasks.

Special Characters or Symbols

Dot matrix printing lends itself well to the creation of custom characters and symbols. There are two aspects to implementing special characters. First, a character look-up table, and second, additional software for dec oding and processing the special characters or symbols. Special characters might be scientific notation, mathematical symbols, unique language characters, or block and line graphics characters.

The character look-up table could be an additional page of Program Memory dedicated to the special characters, or replace part, or all, of the existing look-up tables. If an additional look-up table is used, a third page test would be needed at the beginning of the Character Translation subroutine. There is fundamentally no difference between the processing of special characters and standard ASCII printable characters. If the characters require the same 5 x 7 dot matrix, the balance of the software would remain the same. If, however, the special characters require a different matrix, or the manipulation of the matrix, the software becomes more complex.

In general, the major software modification required to implement special characters is the size of the dot matrix printed or the dot matrix configuration used. In the case of scientific characters, it would often be necessary to shift the 5 x 7 matrix pattern within the available 9 x 9 matrix. Block or line graphics characters, on-the-other-hand, would require using the entire 9 x 9 print head matrix and printing during normally blank dot columns. This would require suspending the blank column blanking mechanism implemented in this Application Note. This would be the most complex aspect of implementing special characters. It would possibly change the number of required instructions, and thus the timing between PTS detection and print head solenoid trigger firing. This could cause the dot columns to be misaligned within a printed line and between lines.

In the case of a matrix change, two approaches are possible: dynamically changing the matrix, in line, as standard ASCII characters are being printed, or isolating the special characters to a separate processing flow where special characters are handled as a unique and complete line of characters only. A discussion of in line matrix changes for special characters is beyond the scope of this Appendix. It is sufficient to say that the changes would require the conditions setting the EOLN flag, character count, and dot column count software be modified during character processing and printing.

Lower Case Descenders

The general principle of implementing lower case descenders is to shift the 5 x 7 character dot matrix within the available 9 x 9 print head solenoid matrix. Implementing lower case descenders requires two software modifications and the creation of status flag for the purpose. First, the detection of characters needing descenders and setting a dedicated status flag during the character code to dot pattern translation subroutine. Second, the character dot column data output to the print head solenoids must be shifted for each dot column of the character. At the end of the character, the flag would be reset.

Inline Control Codes

Inline control codes are two to three character sequences, which indicate special hardware conditions or software flow control and branching. The first character indicates that the control code sequence is beginning and is typically an ASCII Escape character (ESC), 1BH. Termination of the inline code sequence would be indicated by a default number of code sequence characters. This would decrease the buffer size available for characters. Full 80 character line buffering would require loading the Character Buffer with a received character as a character is removed from it and processed.

The Inline Control Code test would be performed in two places: in the Character Buffer Fill subroutine and in the Character Processing (translation) subroutine. The test would be performed in the same manner that a Carriage Return (CR) character code test is implemented. Examples are horizontal tabs and expanded or condensed character fonts. In the case of horizontal tabs, 20H (Space Character) would have to be placed in the Character Buffer for inline processing during character processing and printing. Unless fixed position tabs are used, a minimum of a nibble of Data Memory would be required to maintain a "spaces-to-tab" count. Fixed tab positions could be set via another inline control code, by default of the printer software, or through the use of external hardware switch settings. The control code method of setting the tab positions is the most desireable, but the most complex to implement.

Different Character Formats

Figure B1 illustrates three different character fonts; standard, condensed, and enlarged or expanded characters. As the the figure illustrates, condensed and

enlarged characters are variations in either the number of dots and/or the space used to print them. Thus, each character is a variation of the stepper motor and/or print head solenoid trigger timings. Figure B2 illustrates the timings required to implement the additional character printing.

In addition to the three character fonts shown, it is possible to print each in bold face by printing each dot twice per dot column position. This would require little software modification, but would require a status flag. Again, care must be used to ensure that the delay in retriggering the solenoids is precisely the same for each type of event. Without this precise timing the dot column alignment will not be accurate. The software modifications needed to implement enlarged or condensed characters is essentially the same. The carriage and print head solenoid firing software flow is the same, but the timing for each changes. For condensed characters, the step Time Constant is doubled to approximately 4.08 ms, and the solenoids are fired four times within each step time. The step rate actually becomes a multiple of the solenoid firing time, and a counter incrementing once for each solenoid firing would be needed. At the count of four, the carriage stepper motor is stepped and the counter reset.

In the case of condensed characters, PTS does not play the same roll as in standard or enlarged character printing. PTS is not used to indicate the optimum print head solenoid firing time. Solenoid firing is purely a time function for condensed characters. PTS would only be used for Failsafe protection.

Enlarged characters would require the solenoids be fired twice per dot column data, in two sequential dot columns, at the same rate as standard characters. The character dot column data and dot column count would not be incremented at each output but at every other output. A flag could be used for this purpose.

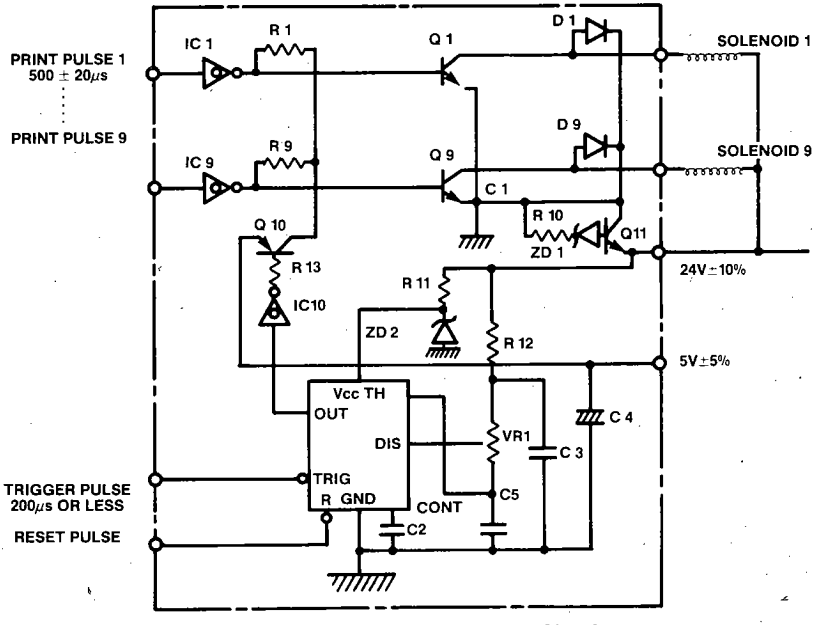
When printing either condensed or enlarged characters, the maximum character count would have to compensate for the increased or decreased characters per line count. When printing enlarged characters, the maximum characters per line would be 40. The Character Buffer could hold two complete lines of characters. But, condensed characters presents a quite different situation. The available character per line increases to 132, well beyond the 80 character Character Buffer size. The solution is to re-initialize the Character Buffer Size Count register count during condensed character processing. This will effectively inhibit the carriage stepper motor drive EOLN detection.

Two status flags would be required; one for standard or enlarged characters, and the second for condensed characters. A third status flag would be required to implement bold face printing. Activating one of the alternate character fonts could be either through the use of external status switches or through inline control code sequences, as detailed above. Note, that if the alternate character fonts are implemented in such a way that format changing is to occur dynamically during any single line being printed, the same control code problems described above also apply. In addition, the effect on the timing and dot column alignment must also be investigated.

Variable Line Spacing

Variable line spacing is another feature which could be implemented either through the use of external status switches or inline control codes. The line spacing is a function of the number of steps the stepper motor rotates for a given line. Figure 15, Paper Feed Stepper Motor Predetermined Time Constants, in the Background section above, lists the Time Constants required for three different line spacings; 6, 8, and 10 lines per inch. At the beginning of the Paper Feed Stepper Motor Drive subroutine, the default line step count is loaded. The software required is a conditional load for the line spacing, indicated by a status flag set in the External Status Switch Check subroutine or the Character Buffer Fill subroutine. Implementing the three different line spacings would require two additional status flags.

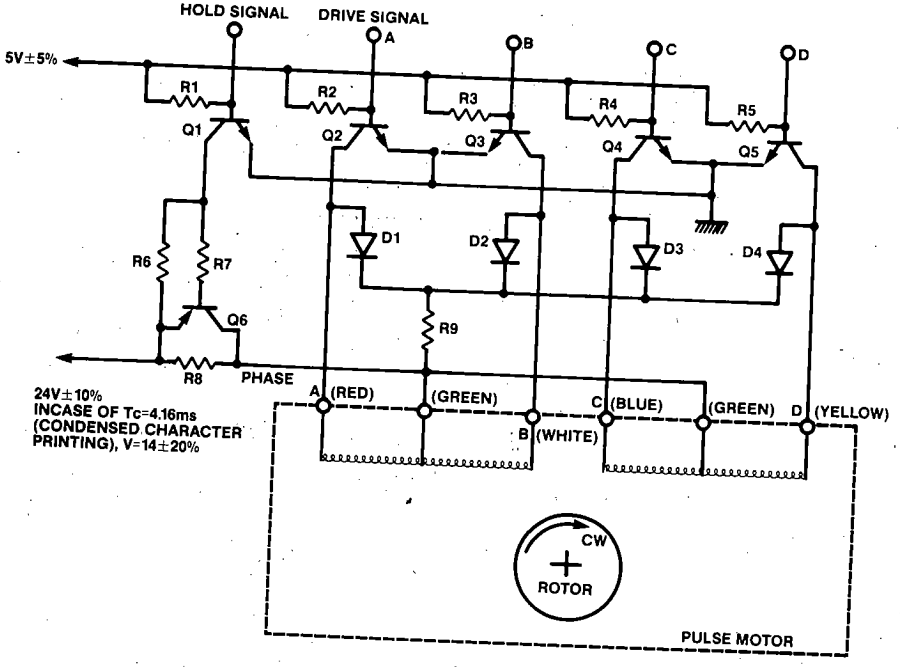
APPENDIX C.
PRINTER MECHANISM DRIVE CIRCUIT



Recommended Solenoid Drive Circuit

PARTS NO.		TYPE	MAKER
IC1~IC10		SN7406	TI
IC11		μA555	Fairchild
D1~D9	DIODE	S5277B	Toshiba
Q1~Q9	TRANSISTOR	2SD986	NEC
Q10	TRANSISTOR	2SA1015	Toshiba
Q11	TRANSISTOR	2SD633	Toshiba
R1~R9	RESISTOR	1.2kΩ ¼	
R10	RESISTOR	22Ω ¼	
R11	RESISTOR	580Ω 2	
R12	RESISTOR	15kΩ ¼ Carbon fil=	
R13	RESISTOR	1.2kΩ ¼	
VR1	VARIABLE RESISTOR	20kΩ ¼	
C1	CAPACITOR	1μF 100V	
C2	CAPACITOR	0.01μF	
C3	CAPACITOR	0.001μF	
C4	CAPACITOR	10μF 16V	
C5	CAPACITOR	0.1μF fil=	
ZD1	ZENOR DIODE	HZ24	Hitachi
ZD2	ZENOR DIODE	HZ5C1	Hitachi

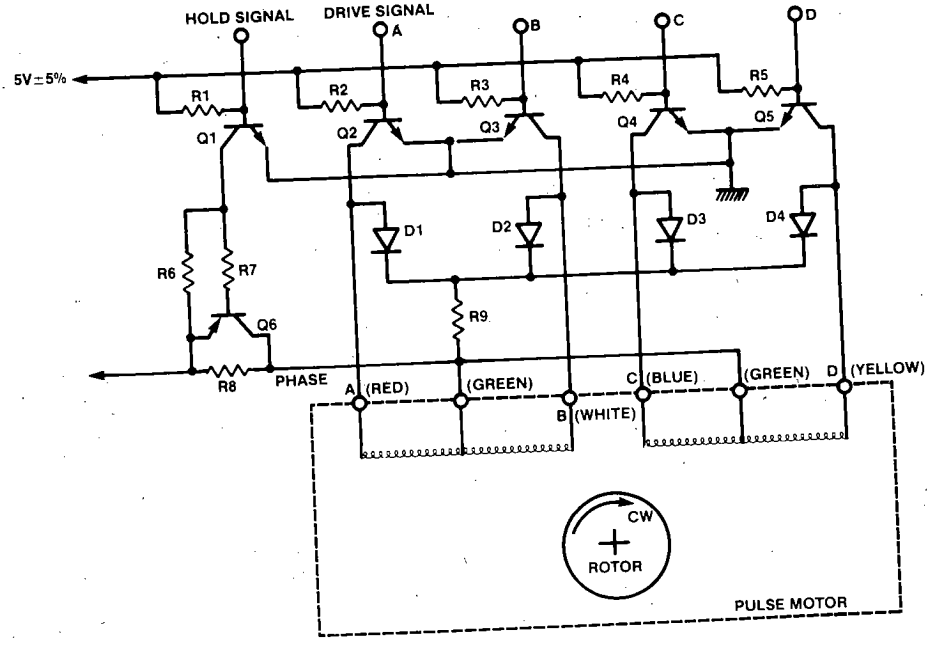
Recommended Carriage Motor Drive Circuit



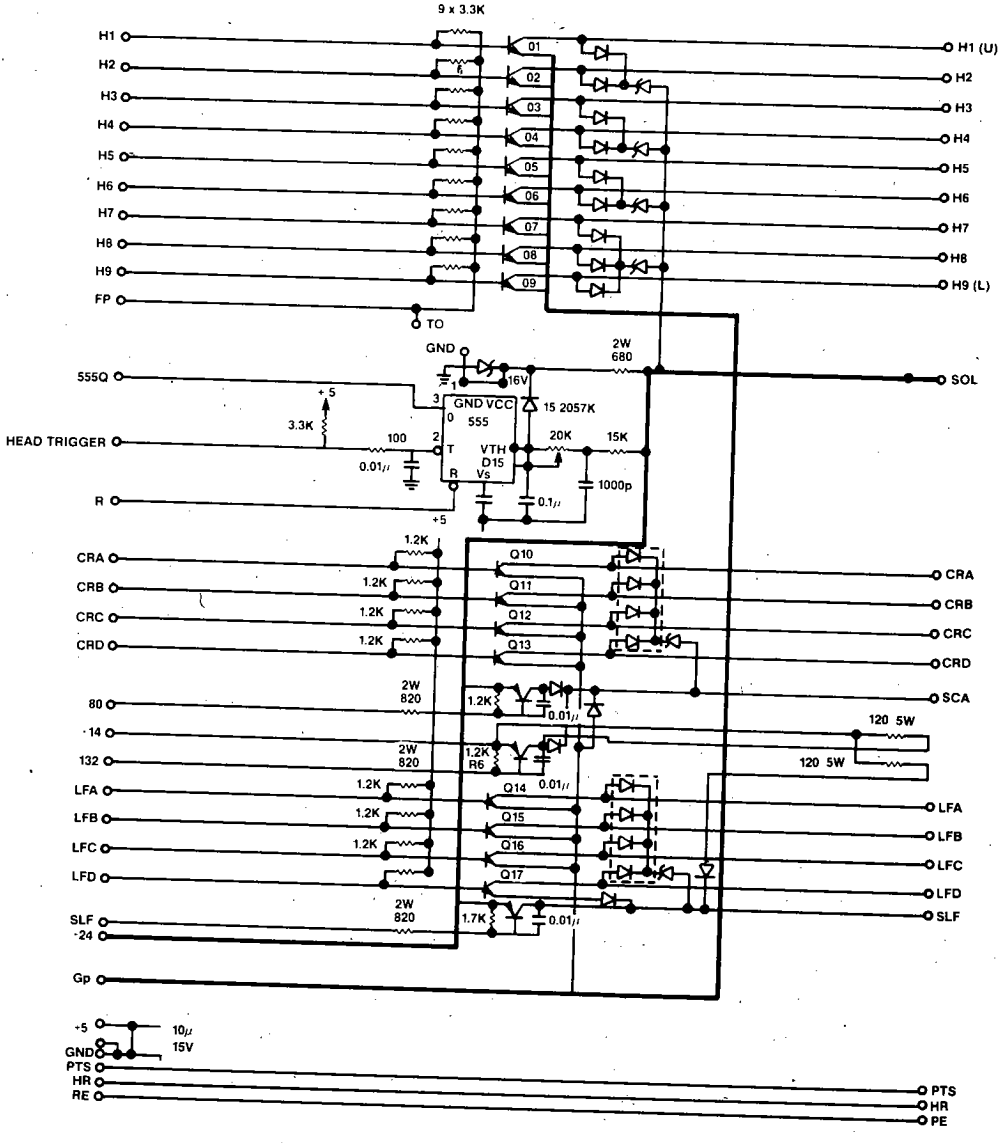
PARTS NO.		TYPE	MAKER	QTY
R1	Resistor	1kΩ±10% ¼		1
R2-R5	Resistor	220Ω±10% ¼		4
R6	Resistor	10kΩ±10% ¼		1
R7	Resistor	470Ω±10% 3		1
R8	Resistor	130Ω±10% 7		1
R9	Resistor	330Ω±10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2~Q5	Transistor	2SD526—Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1~D4	Diode	1S954	NEC	4

230795-001

Recommended Paper Feed Motor Drive Circuit



PARTS NO.		TYPE	MAKER	QTY
R1	Resistor	1kΩ±10% ¼		1
R2-R5	Resistor	220Ω±10% ¼		4
R6	Resistor	10kΩ±10% ¼		1
R7	Resistor	470Ω±10% 3		1
R8	Resistor	130Ω±10% 7		1
R9	Resistor	330Ω±10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2~Q5	Transistor	2SD526—Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1~D4	Diode	1S954	NEC	4


9 x 3.3K
H1
H2
H3
H4
H5
H6
H7
H8
H9
FP
TO
01
02
03
04
05
06
07
08
09
H1 (U)
H2
H3
H4
H5
H6
H7
H8
H9 (L)
GND
2W
680
SOL
555Q
16V
15 2057K
GND VCC
555
3.3K
+5
HEAD TRIGGER
100
0.01μ
VTH
D15
Vs
20K
15K
1000p
0.1μ
R
CRA
CRB
CRC
CRD
1.2K
Q10
Q11
Q12
Q13
2W
820
80
14
132
LFA
LFB
LFC
LFD
SLF
24
Gp
R6
0.01μ
Q14
Q15
Q16
Q17
1.7K
120 5W
SCA
10μ
15V
PTS
HR
RE
PE

230795-001